SAS® GLOBAL FORUM 2021

1135-2021

Scaffolding SAS®Projects With NPM and SASjs

Allan Bowe

ABSTRACT

Is your SAS scattered across SAS folders, local directories and server file systems? Are your deliverables delayed by the need to build tools? Does your deployment process involve manual steps?

The SASjs framework enables code consistency across teams and projects, de-risks the use of shared tools and dependencies, and facilitates continuous deployment to SAS environments. The framework can be applied to SAS® Viya® jobs, SAS® 9 Stored Processes, and even regular SAS programs on a file system.

Join this session and learn how to create (scaffold) a SASjs project, add a job, add a macro dependency, add a program (include) dependency, add job init and term files, deploy the jobs to SAS, and run them as part of a flow. I will share the secret of how to deploy to SAS Viya without a client and secret. And, the entire demo will be performed from a local text editor (VSCode).

INTRODUCTION

If you are a SAS Developer with two or more projects under your belt, you'll be familiar with the fact that SAS affords incredible flexibility when it comes to development and deployment. Every company has different standards, and even you yourself have probably developed some coding habits that are not shared by your colleagues.

As it happens, this same situation applies also to JS (JavaScript) - a flexible and loosely typed language, often mis-used to create spaghetti projects that are hard to maintain and difficult to debug.

In response to this situation, various frameworks have evolved that 'wrap around' JS to provide consistency and structure in the way projects are built - such as React, Angular, and Vue. This is great for application owners, making it easier to on-board new developers, and reducing the total cost of ownership, whilst maintaining development velocity as application complexity increases.

The goal of SASjs is to provide such a framework for SAS - a structure, set of opinions, and suite of productivity tools. This enables developers to spend more time on the things that matter - ie, rapid delivery of business value, to any SAS platform, in a repeatable and scalable fashion.

In this paper we walk through the process of setting up a SASjs project in terms of:

- deployment
- execution
- documentation
- testing

THE SASJS FRAMEWORK

The SASjs Framework itself is a set of tools that abstracts away the common complexities of a typical SAS deployment. Each of these tools can be used individually, or together as part of a cohesive and integrated approach to managing the SAS application lifecycle - ie project setup, development, compilation, deployment, execution, testing, documentation, and more.

The core components are:

- sasjs/adapter - handles SASLogon authentication and communication between frontend and backend
- sasjs/cli - a Command Line Interface for automating common tasks
- sasjs/core - a library with over 120 SAS macros geared towards application developers

Other components include:

- sasjs/vscode-extension - SAS code execution, syntax highlighting and linting in the VS Code IDE
- sasjs/lint - the linter is used in both the VS Code extension and the SASjs CLI
- numerous seed apps ready to kick start a SAS project

There are even a series of ready-made apps, built using the framework.

GETTING STARTED

There are some pre-requisites to install before continuing:

- NPM. This provides the run-time for the CLI. Instructions for the portable version are here.
- GIT. Not strictly necessary but highly recommended, and git-bash provides unix-like commands in windows.
- VS Code. You could use other IDEs, however this is the one primarily supported by SASjs.

For most of the rest of this paper we will be submitting terminal commands. If you are working in VS Code you may wish to change your default terminal to git-bash - to do this, open Terminal (Terminal/New Terminal), choose "select default profile" and select "git bash" from the list of options.

To check you have everything installed correctly, try running:

```
npm -v
```

This should return a version number. If not, go back and check your installation of NodeJS (did you add the correct folder to your PATH if running the portable version? If in VS Code, did you try re-opening the editor?)

If you have a version number, you are ready to install the SASjs CLI. Simply run:

```
npm install -g @sasjs/cli
```

This will install the SASjs CLI globally, such that you can run the `sasjs` command from any directory. You can verify this by running:

```
sasjs v
```

This will return the version number. So - now that we have the CLI - we can get properly started!

SASJS CREATE

This command is used to create an initial project, from a template. We'll use the jobs template. Run the following commands to create a project called "sgf2021" and move into that directory.

```
sasjs create sgf2021 -t jobs
cd sgf2021
```

Here you will see a number of files/folders:

File/Folder	Description
db	This folder contains the DDL for creating the various libraries. It is split with one subfolder per libref.
.git	System folder for managing GIT history
.github	Contains example github actions should you wish to perform automated SASjs checks, eg as part of a pull request or merge
.gitignore	List of files and folders to be ignored by GIT (won't be committed to source control)
.gitpod.dockerfile	Used for gitpod demos, can be ignored / removed
.gitpod.yml	Used for gitpod demos, can be ignored / removed
includes	Default location for SAS Programs, to be `%include`d by SAS Jobs, Services, or Tests
jobs	Main parent folder containing `jobinit.sas` and `jobterm.sas` as well as subdirectories for SAS Jobs
macros	Default location for SAS Macros
node_modules	System folder for storing third party packages, such as @sasjs/core
package.json	Main config file for the repo, lists all the project dependencies as well as example scripts (npm run SCRIPTNAME) and project metadata

File/Folder	Description
package-lock.json	System file for managing dependencies of dependencies
README.md	The main page describing your project. Is also used as the homepage when running `sasjs doc`.
sasjs	The folder containing the `sasjsconfig.json` file, which contains the settings for the entire project - such as where the various files are located (both locally and remotely).
.sasjslint	Settings used by `sasjs lint` to check SAS Jobs, Services, Tests, Macros and Programs for code quality issues such as encoded passwords, trailing spaces, macros without parentheses, etc etc.
tests	Alternative location for storing tests. Normally we recommend that tests are stored alongside the relevant Job/Service/Macro, eg `jobname.test.sas` or `macroname.test.sas`
.vscode	Folder containing VSCode dependencies, such as the default 80 char ruler, and the recommendation to use the SASjs extension.

This project is already 'ready to build' however to explain the process, let's add a new job.

Adding a Job

To add a job, simply add an "any_name_you_like.sas" file to any of the folders listed in the "sasjs/sasjsconfig.json" `jobFolders` array. You could create a new folder to put it in (listed in this array) or use an existing one - such as jobs/load.

All SAS files in SASjs **must** have a Doxygen header. This enables the automatic documentation which we'll cover in `sasjs doc`.

The header looks like this:

```
/**
  @file
  @brief one line description of the job
  @details Multiline description of the job.
  You can write markdown here and it will be properly
  rendered in the HTML docs.

      data code must be;
        indented = 4 * spaces;
      run;

  <h4> SAS Macros </h4>
  @li example.sas

  <h4> SAS Includes </h4>
  @li someprogram.sas MYREF

  <h4> Data Inputs </h4>
  @li somelib.ds1
  @li somelib.ds2
  <h4> Data Outputs </h4>
  @li tgtlib.bigmuvvads

**/
```

For a longer, better description of Doxygen, please see Paper 1077-2021 (Introduction to Doxygen) by Tom Bellmer.

There are some additional sections in this header that must be observed. The Data Inputs / Outputs section is used by `sasjs doc` to generate a graphviz diagram of the data lineage.

The SAS Macros / SAS Includes sections are used by `sasjs compile` to identify, and then insert, Macros and Programs at the beginning of each compiled job.

Feel free to add some SAS code at the bottom of your new job! Perhaps even commit it to a GIT repository.

That's as far as we'll go with modifying the repo - the rest of the paper is about compiling, building, depoying, testing & documenting the code.

SASJS COMPILE

The compilation process is about taking all of the dependencies (SAS Macros & Includes) and creating **one file per Job/Service/Test**.

Why do this? Isn't this against the philosophy of having a single macro definition, in a catalog or SASAUTOS fileref, and using it everywhere?

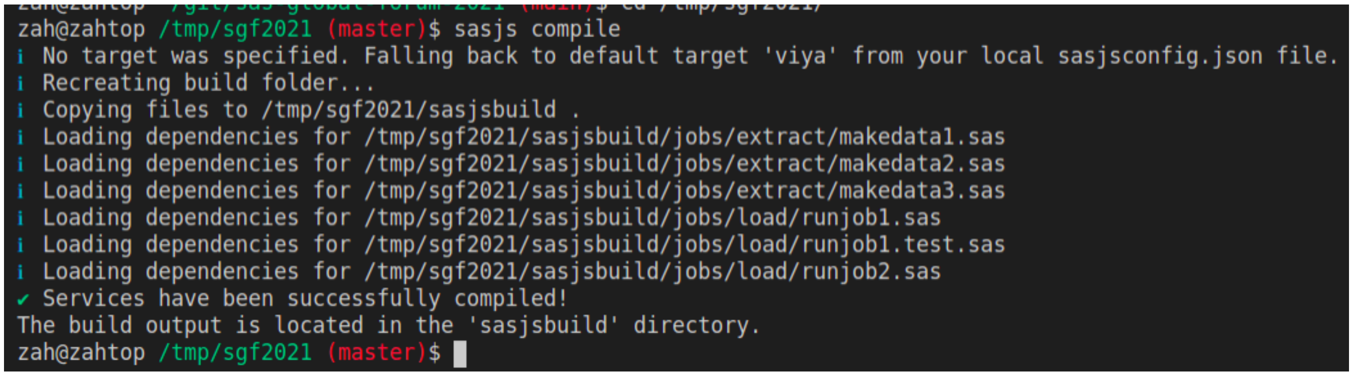
Maybe this was a good idea back in the days when disk space was scarce, and developers worked without source control. These days, thanks to tools like GIT, it's very easy to track changes to a file. And with a move to container based runtimes, like Viya 4, there is a need to move away from OS filesystems. Finally, if you have ever tried to debug a complex macro application, you'll quickly discover how challenging it can be to unearth all the underlying macros from a monolothic enterprise estate.

The SASjs approach is to compile ALL dependencies so each Job / Test / Service is **fully self contained**, and can live inside the metadata of a Stored Process or a Viya Job description. The impact on runtimes and storage is negligible, more than compensated by the ease of debugging and the portability of the jobs. In addition, it becomes easy to track which jobs have which macros (as they must be listed in the program header in order to be compiled).

To execute the compilation process then, simply run:

```
sasjs compile
```

This takes all the jobs/services/tests and creates the self-contained files inside the (temporary) "sasjsbuild" folder.



```
zah@zahtop /tmp/sgf2021 (master)$ sasjs compile
ℹ No target was specified. Falling back to default target 'viya' from your local sasjsconfig.json file.
ℹ Recreating build folder...
ℹ Copying files to /tmp/sgf2021/sasjsbuild .
ℹ Loading dependencies for /tmp/sgf2021/sasjsbuild/jobs/extract/makedata1.sas
ℹ Loading dependencies for /tmp/sgf2021/sasjsbuild/jobs/extract/makedata2.sas
ℹ Loading dependencies for /tmp/sgf2021/sasjsbuild/jobs/extract/makedata3.sas
ℹ Loading dependencies for /tmp/sgf2021/sasjsbuild/jobs/load/runjob1.sas
ℹ Loading dependencies for /tmp/sgf2021/sasjsbuild/jobs/load/runjob1.test.sas
ℹ Loading dependencies for /tmp/sgf2021/sasjsbuild/jobs/load/runjob2.sas
✔ Services have been successfully compiled!
The build output is located in the 'sasjsbuild' directory.
zah@zahtop /tmp/sgf2021 (master)$
```

Figure 1: sasjs compile command

Whilst SAS Macros (and in addition, the jobinit.sas / serviceinit.sas files) can be easily inserted into the beginning of the Job/Sevice, the SAS Includes are a bit trickier to manage - if SAS code is simply inserted into a Job, it is then executed at the start, which isn't that useful.

For this reason, the SAS Includes are first wrapped by `sasjs compile` into `put` statements against a user-designated fileref - where they can be subsequently `%include`'d.

So, by inserting a header section in the following format:

```
<h4> SAS Includes </h4>
@li demotable_ddl.sas FREF1
```

The final, compiled, code will contain an initial section like this:

```
filename FREF1 temp;
data _null_;
file FREF1 lrecl=32767;
put '/**';
put '  @file';
put '  @brief DDL for demotable1';
put '**/';
put 'proc sql;';
put 'create table &mylib..demotable1(';
put '        tx_from num not null format=datetime19.3';
put '        ,tx_to num not null format=datetime19.3';
put '        ,vara varchar(10) not null';
put '        ,varb varchar(32) not null';
put '    ,constraint pk_demotable1';
put '        primary key(tx_from, vara));';
run;
```

That program can then be easily invoked anywhere in the actual job with a single line of code:

```
%inc fref1;
```

The following diagram illustrates this concept, looking specifically at Services and Macros (the idea is the same for Jobs and Tests, with both Macros and Includes):

More info here: https://cli.sasjs.io/compile

What next? We could have hundreds of these compiled files. How do we push them all up to the SAS server?

SASJS BUILD

Given that we can have so many files, the next step is to create a portable "build" pack to make it much easier to perform the subsequent deployment.

Running `sasjs build` will actually create two files:

- A JSON file, used to deploy Jobs to Viya using the APIs
- A SAS program, which can be executed to create Jobs/Services/Tests in both SAS 9 or SAS Viya.

How do we know whether to generate a SAS Program to create Stored Processes (SAS 9) or Jobs (SAS Viya)?

And in which (logical) folder will they be created?

The answer to these questions is contained in the **target**. The target is defined in the `sasjs/sasjsconfig.json` file and contains (at a minimum):

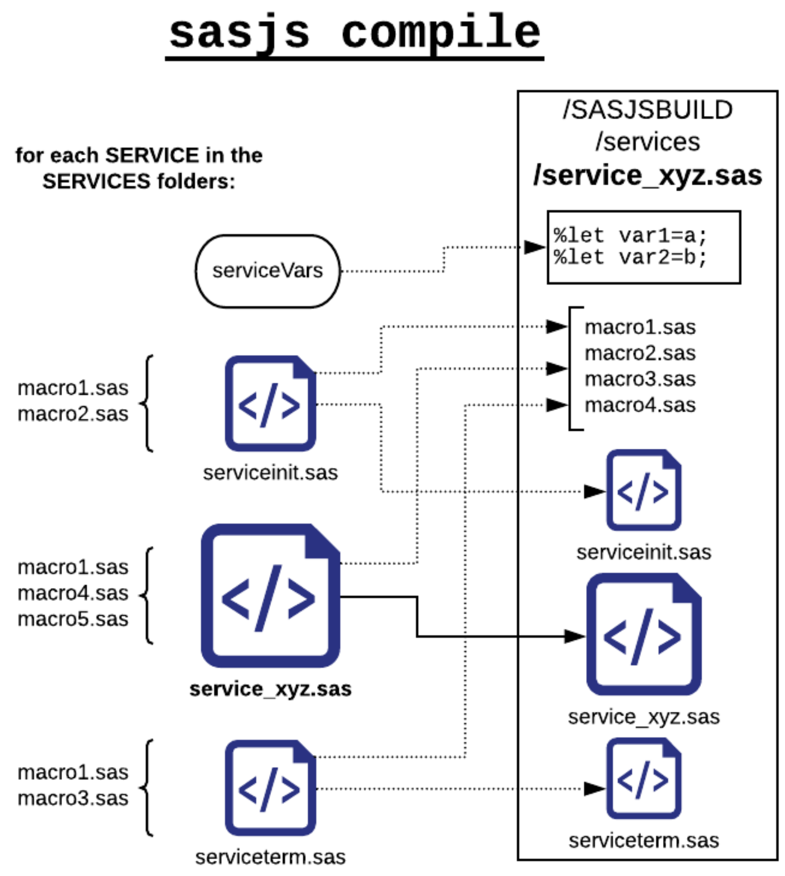


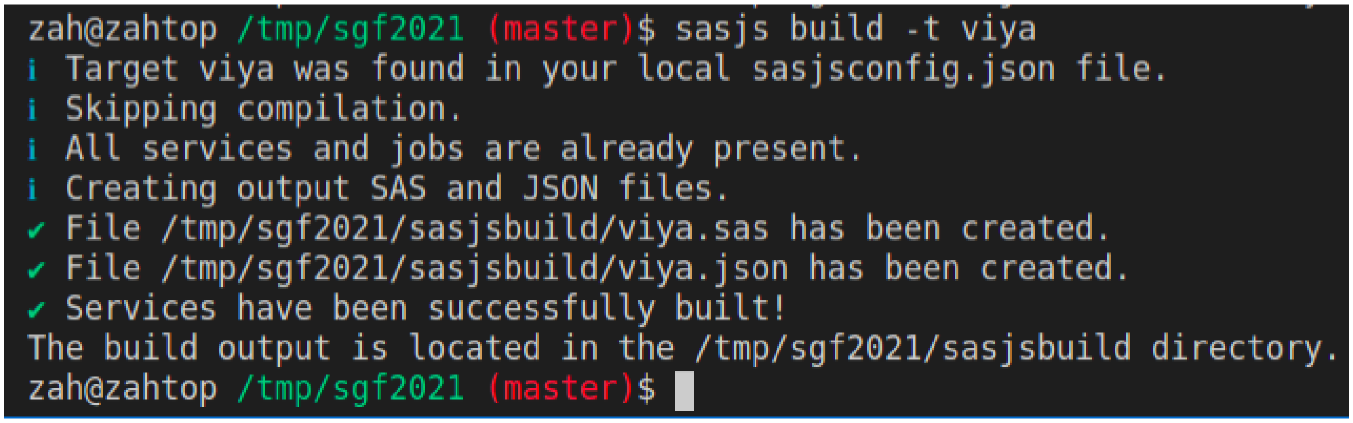
Figure 2: sasjs compile

- Target `name`. This is a one-word alias, used in the sasjs commands to refer to the target definition.
- `appLoc`. The root folder in SAS Metadata or SAS Drive under which our content will be deployed.
- `serverUrl`. This is the server on which the assets will be deployed
- `servertype`. Either SAS9 or SASVIYA, this attribute tells `sasjs build` which type of program to create.

Targets can contain additional attributes, such as target-specfic jobs, tests, macros or services.

To create our build back then, on the target named “viya”, we will run:

```
sasjs build -t viya
```



```
zah@zahtop /tmp/sgf2021 (master)$ sasjs build -t viya
i Target viya was found in your local sasjsconfig.json file.
i Skipping compilation.
i All services and jobs are already present.
i Creating output SAS and JSON files.
✔ File /tmp/sgf2021/sasjsbuild/viya.sas has been created.
✔ File /tmp/sgf2021/sasjsbuild/viya.json has been created.
✔ Services have been successfully built!
The build output is located in the /tmp/sgf2021/sasjsbuild directory.
zah@zahtop /tmp/sgf2021 (master)$
```

Figure 3: sasjs build console window

More info here: https://cli.sasjs.io/build

So, we have generated our build program and JSON file in the root of the `sasjsbuild` folder. How to get that up to the server?

SASJS DEPLOY

The actual deployment process will vary depending on the server and level of access you have. Our demo will focus on using Viya REST APIs, but if you are on SAS 9 then you have two options available:

1) Run the SAS program in Enterprise Guide or SAS Studio (you can of course also do this in Viya, without the need for a client/secret)
2) Use CURL as part of a bash script in your `deployScripts` array to be `%include`d as part of an STP in your home directory

We are working on a more integrated way to manage this, possibly using the SAS 9 REST API to create the STPs directly. Get in touch if you’d like to engage us to push this forward.

For a Viya deploy, we are going to need a client/secret pair.

Generating a Viya Token

A client/secret pair is necessary to grant the SASjs CLI permission to deploy to Viya on your behalf. It’s sensible therefore to grant the minimum permissions possible. And also to ensure that you NEVER commit the resulting ACCESS / REFRESH token to GIT.

We take the following security precautions:

- Basic (user/pass) authentication is not supported. Use authorization_code grant type when creating the client/secret.

- Tokens are stored in a `.env` file that is included in `.gitignore` to prevent accidental commits

To generate your client/secret you will need the help of an administrator, who has the filesystem permissions to access the consul token.

We have built the Viya Token Generator web app to make this proces super easy.

You can also generate a client / secret (and access / refresh token) using SAS code using the [@sasjs/core](https://core.sasjs.io) macros.

```
/* compile the macros from github */
filename mc url "https://raw.githubusercontent.com/sasjs/core/main/all.sas";
%inc mc;

/* create random client and secret */
%mv_registerclient(outds=clientinfo)
```

Ok, you have your client & secret. Now what? Don't procrastinate. Authenticate!

SASjs ADD CRED

We need to assign credentials to the particular target (server details) to which we are deploying. This makes sense if you consider that you could be deploying to multiple machines (dev / test /prod) or server types (sas9 / viya).

The following command will give you a series of prompts to authenticate against the "viya" target - you can now enter your client / secret / serverUrl, click the URL to fetch the authorisation code, paste, and finally choose a Compute Context on which you'd like to perform operations.

```
sasjs add cred -t viya
```

Your credentials will be added to a `.env.viya` file. If you have multiple targets and would like to re-use credentials, just rename this file to `.env` (it'll be a catch all if a target specific file is not found).

DOing the DEPLOY

We're finally ready for blast off. Run the following to deploy all your jobs / services / tests to the designated appLoc in the "viya" target:

```
sasjs deploy -t viya
```

You can also perform all three of the previous steps (compile, build, deploy) in just one short command:

```
sajs cbd
```

Well done! This is the first step in an iterative, GIT-centric, dev-ops orientated coding paradigm for SAS.

You can work safely (locally) in a feature branch, deploy your code to a personal `appLoc`, execute and test your code, before finally merging to a development or main/master branch.

More details here: https://cli.sasjs.io/deploy

So - what else is there?

SASJS JOB EXECUTE

The benefit of using SASjs to execute your SAS jobs is mainly that you can store and browse the logs locally. An example command looks like so:

```
sasjs job execute jobs/extract/makedata1 -l sasjsbuild/makedata1.log -t viya
```

The job path can be relative (to the appLoc) or a full path. The log path can also be relative (to the sasjs folder) or a full path.

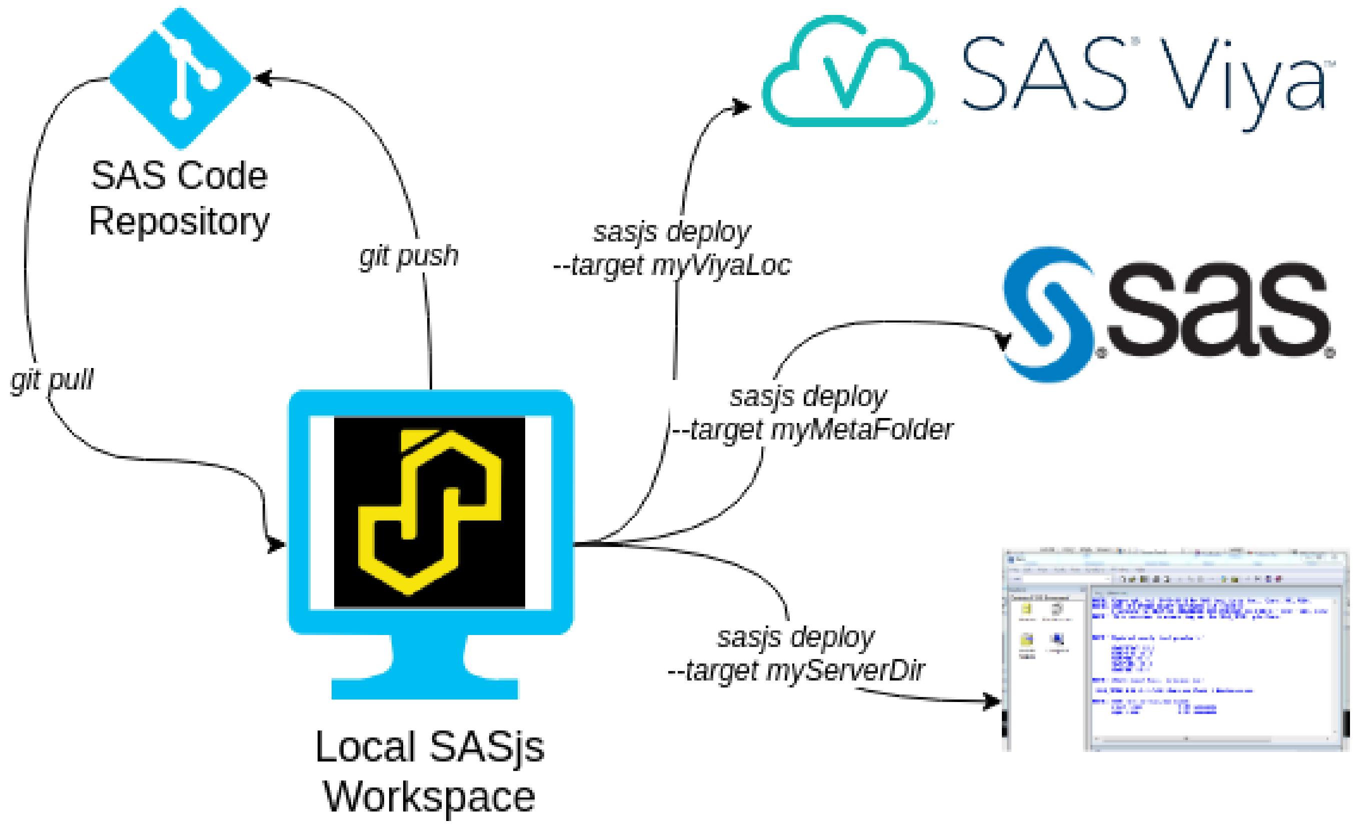

SAS Code Repository
git push
git pull
sasjs deploy --target myViyaLoc
SAS Viya
sasjs deploy --target myMetaFolder
sas
sasjs deploy --target myServerDir
Local SASjs Workspace

Figure 4: sasjs deploy workflow

More details here: https://cli.sasjs.io/job

SASJS FLOW EXECUTE

You can define a series of jobs, in waves, with dependencies, as part of an input JSON file. If the job has errors or warnings, subsequent waves will terminate. All logs are stored locally, and a CSV file is created showing the job results. The PID is also identified for each running job.

Example invocation:

```
sasjs flow execute -s sasjs/jobflow.json -l sasjsbuild --csvFile sasjsbuild/results.csv -t viya
```

More details here: https://cli.sasjs.io/flow

SASJS DOC

The great benefit of using Doxygen is the ability to generate documentation for Jobs, Services, Macros, Program Includes, and Tests.

The `sasjs doc` command will take metadata from the package.json file, use the README.md as a homepage, and even generate a data lineage graphviz diagram using the defined “Data Inputs” and “Data Outputs” in your program headers. Clicking on a job will open the job in the documentation, and you can even click through the tables into the Data Controller for SAS® Data Viewer.

The logo and other such settings can all be configured in the docConfig.

To run the command, simply execute:

```
sasjs doc
```

The outputs will go into the sasjsbuild/docs folder.

More info here: https://cli.sasjs.io/doc

SASJS TEST

The latest addition to the SASjs Framework is SASjs Test. At the time of writing this part isn’t *actually* live yet... However it’s very close!!! So I’ve written the paper above as though it already exists. Here is the pull request.

The aim in building a test framework is to make it very easy to write tests for Jobs, Services and Macros. With this in mind, all you need to do, is create a same named test file named `yourjobname.test.sas`. You are recommended to do this in the same folder as the original Job / Service. Any files suffixed `.test.sas` will be compiled and deployed to a seperate `appLoc/tests/xxx` subfolder.

Tests will be compiled as web services, so that you can (optionally) return JSON to indicate the success / failure of one or more tests.

Test coverage is determined simply by the existence of at least one file per Job / Service, and details displayed when running `sasjs compile`.

Jobs may be tested by examining the outputs created. Services can be tested by calling them with various inputs, eg using `proc stp` or the mv_jobwaitfor macro.

Additional tests can also be defined in the `testFolders` array. When running `sasjs test`, a `testsetup.sas` file is executed first, then all the tests, and finally a `testteardown.sas`.

The aim is to provide a very simple structure, that is very easy to extend, to encourage more programmers to write tests when developing code.

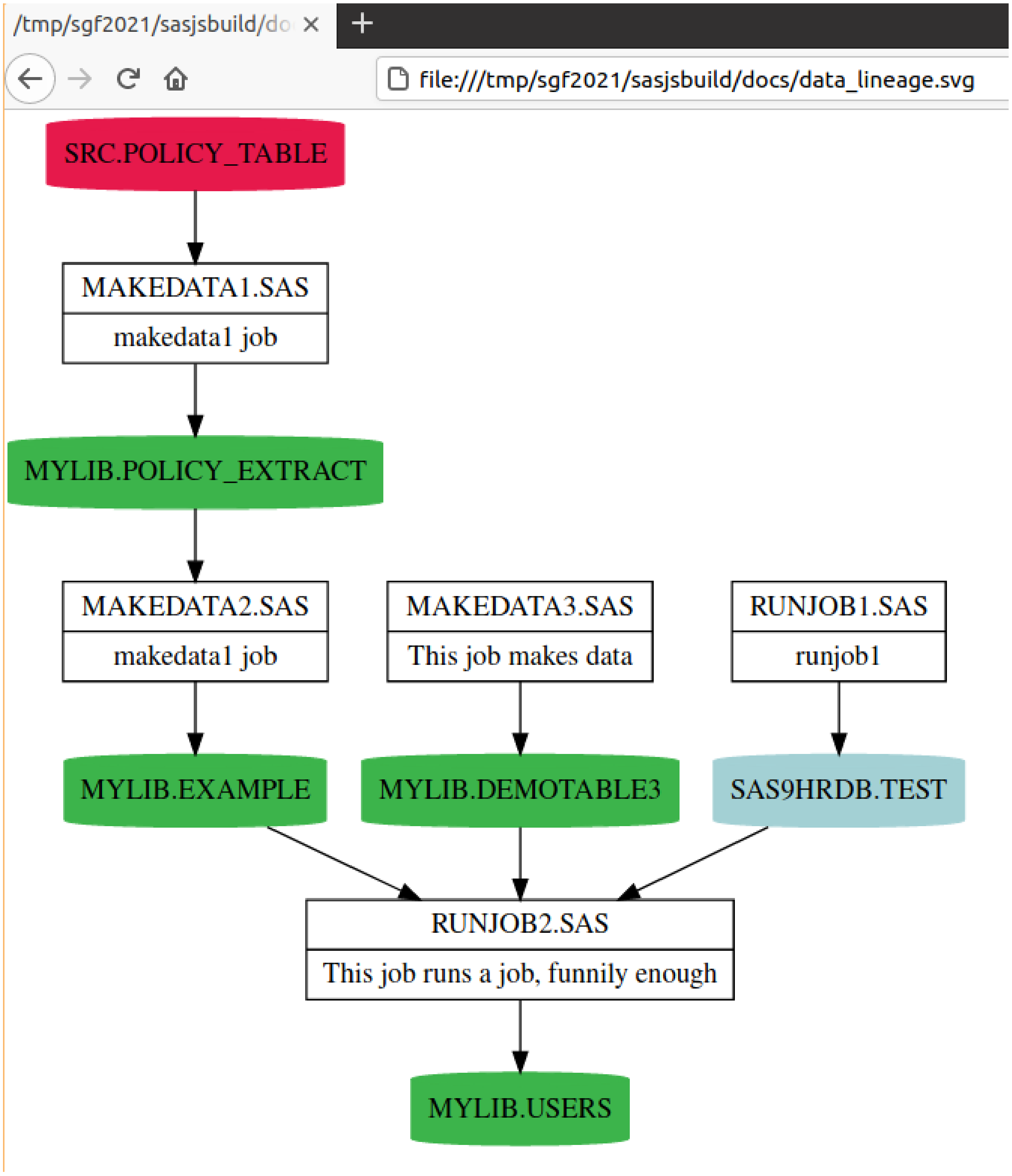
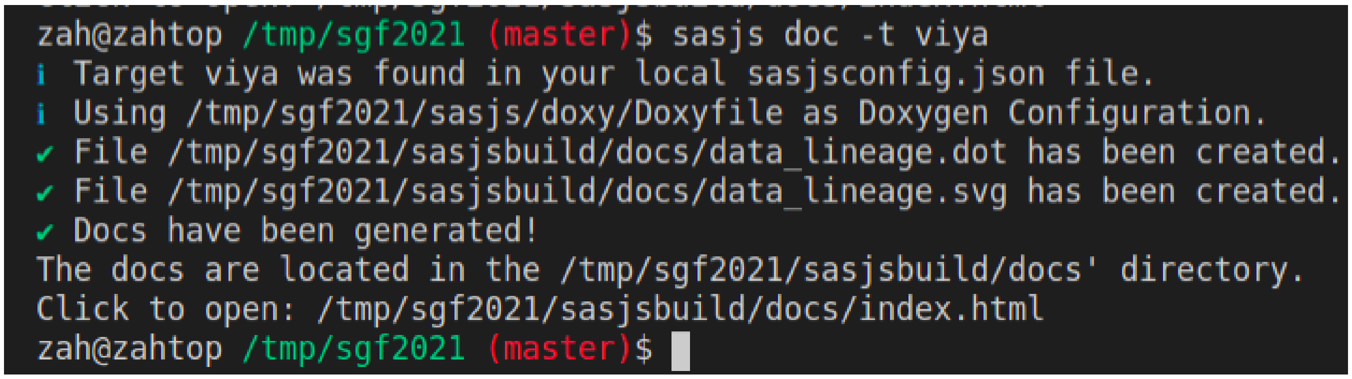


Figure 5: sasjs doc data lineage



```
zah@zahtop /tmp/sgf2021 (master)$ sasjs doc -t viya
i Target viya was found in your local sasjsconfig.json file.
i Using /tmp/sgf2021/sasjs/doxy/Doxyfile as Doxygen Configuration.
✓ File /tmp/sgf2021/sasjsbuild/docs/data_lineage.dot has been created.
✓ File /tmp/sgf2021/sasjsbuild/docs/data_lineage.svg has been created.
✓ Docs have been generated!
The docs are located in the /tmp/sgf2021/sasjsbuild/docs' directory.
Click to open: /tmp/sgf2021/sasjsbuild/docs/index.html
zah@zahtop /tmp/sgf2021 (master)$
```

Figure 6: sasjs doc console window

SASJS LINT

A linter is used to examine source code for common problems, before it is deployed or committed to source control. The SASjs linter is available in the VS Code extension (see “Problems” tab) or as a SASjs CII command.

The great thing about the command (`sasjs lint`) is that you can add it as a pre-commit hook, to prevent questionable code from making it into your repository!

What is questionable? Good question. It can vary from site to site. For that reason, it’s fully configurable. To set up the config you can run `sasjs lint init`. As of April 2021 we have the current settings, but plan to add many more:

```
{
    "noEncodedPasswords": true,
    "hasDoxygenHeader": true,
    "hasMacroNameInMend": true,
    "hasMacroParentheses": true,
    "indentationMultiple": 2,
    "lowerCaseFileNames": true,
    "maxLineLength": 80,
    "noNestedMacros": true,
    "noSpacesInFileNames": true,
    "noTabIndentation": true,
    "noTrailingSpaces": true
}
```

If you’d like to see any specific tests in your SAS code, you are welcome to contribute, or contact us to see if we can put them in for you.

THE FUTURE

Beyond additional functionalities in VS Code (improved linting and formatting) and general framework support, the future for SASjs includes the following major items:

- Full integration with SAS 9 (eg using the SAS 9 REST API)
- NodeJS server for Base SAS (enabling SAS Apps and Dev Ops for traditional SAS users)

If you’d benefit from the above, do get in touch.

CONCLUSION

SASjs was built originally to support Data Controller and the development of Web Apps on SAS. Since then it has matured into an enterprise framework to enable Dev Ops and automated deployment for all types of SAS Platform.

It is fully open source (MIT) and can be used for any purpose without encumbrances or a need for attribution.

Whilst primarily sponsored by Analytium the project itself is independent and fully open for other collaborators. We need help, and additional sponsors, and would love to hear from you!

ACKNOWLEDGMENTS

- Analytium for being the primary project sponsor
- The entire team at SAS Apps for doing the development
- The developers of Doxygen
- All the customers of Data Controller who funded the initial framework

FURTHER READING

- 1124-2021 Building & Deploying Web Apps with SASjs CLI
- 1077-2021 Introduction to Doxygen
- 1072-2021 Open Source SAS® Macros: What? Where? How?

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

- Email: allan.bowe@analytium.co.uk
- LinkedIn: https://www.linkedin.com/in/allanbowe
- Github: https://github.com/allanbowe

LINKS

- Github Organisation: https://github.com/sasjs
- SASjs Main Site: https://sasjs.io
- Macro Core docs: https://core.sasjs.io
- CLI docs: https://cli.sasjs.io
- Adapter docs: https://adapter.sasjs.io
- Data Controller: https://datacontroller.io
- Data Controller docs: https://docs.datacontroller.io
- SAS Apps website: https://sasapps.io