

XMM-Newton SAS Virtual Development Environments

Aitor Ibarra & José Marcos on behalf of XMM-Newton SAS Team

XMM2ATHENA - 26/02/2024

Overview



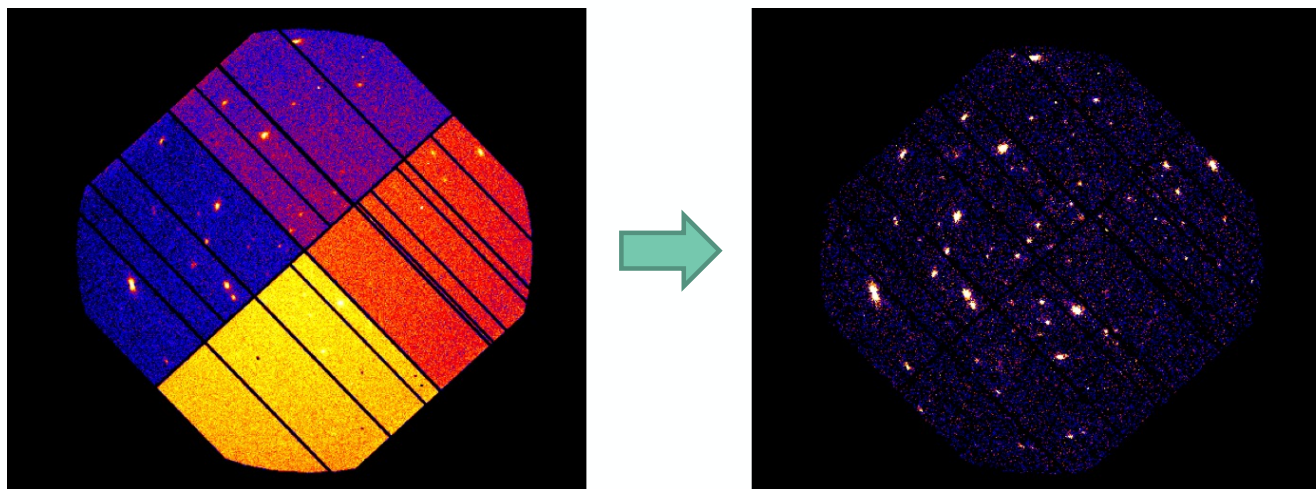
1. XMM-Newton Science Analysis System (SAS) Introduction
2. SAS building process
 1. Infrastructure-as-Code
3. SAS code maintenance
 1. Platform-as-Code
4. SAS Virtual Development Environments
5. SAS on cloud infrastructures: Datalabs
6. What next...



XMM-Newton Science Analysis System (SAS) Introduction



- The XMM-Newton Scientific Analysis System (SAS) is a freely distributed suite of programs for dealing with data from all XMM-Newton instruments.
- SAS is a collection of tasks (C/C++ & Fortran-[77,90]), scripts (perl & python) and libraries, specifically designed to reduce and analyze data collected by the XMM-Newton instruments.
- SAS is able to convert the XMM-Newton data from L0.5 (raw) to L3 (science products)
 - React quickly to new developments in calibration
 - Applies calibrations to raw data



- Optimally screen/filter the data



SAS development process

- SAS code → high dependency on the C/C++/F77/F90 compiler → introduces high dependency on the Operating System
- Third-party libraries makes the building process a bit more complex.
 - HEASOFT → used in harness tests → recommendation build from source code
 - Perl → evolves with time → version dependency
 - TexLive....
- Daily builds based on ftp and tgz
 - Geographically Distributed Code Development.
 - Continuous Integration System BEFORE the Concept was invented!!!
 - Central Code Repository.
 - SAS developers authenticated via gpg key-ring.
- Too complex to maintain this infrastructure and knowledge transfer.

ENVIRONMENT

From configuring build environments manually:



SAS development process: (Dockerized SAS builder)

- Hardware virtualization
 - The first step to reduce costs.
 - But still too manual process.
- Infrastructure as Code
 - New paradigm to encapsulate and automatize the building process.
- Dockerization of the SAS building process:
 - Not easy because of the complexity
- Jenkins as SAS builder orchestrator

Infrastructure as Code:

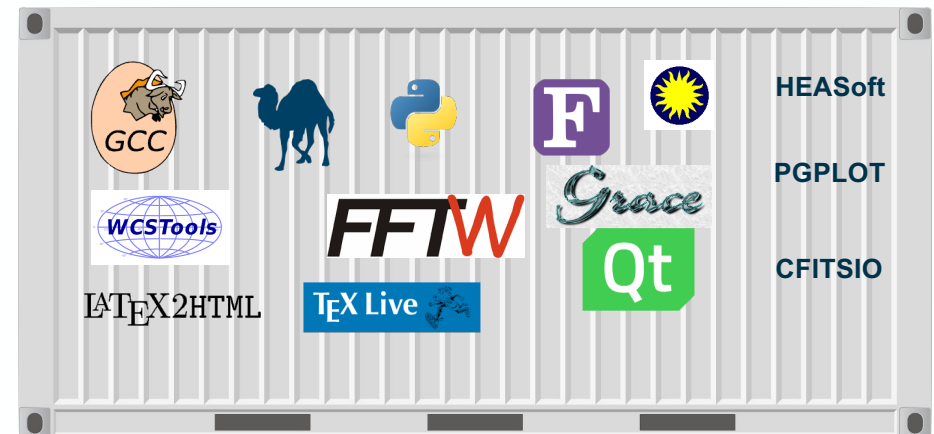
- Dockerfiles
- Bitbucket code repository
- Nexus repository to store third-party libraries versions and SAS docker images

ENVIRONMENT

From configuring build environments manually:



To Docker containers with same tools and configurations:



SAS code maintenance:



- **SAS development takes place on developer's infrastructure**
 - **Coordination with IT team → Different projects – Different needs.**
 - **Infrastructure set-up effort is high.**
 - **Switching between Operating Systems (Linux/Mac OS) to maintain the code is not easy.**
 - **Switching between Operating Systems to work (Windows/Linux) tedious and costly.**
 - **Big organizations tendencies make this process even more complex → ESA365.**

• **SAS Virtual Development Environment**

- **Docker deployment → Platform-as-Code (PaC)**
- **Easy to integrate in Cloud Platforms.**
- **Easy to integrate with common code repositories (Gitlab – Bitbucket)**

laC + PaC

- **Centralized setup and maintenance.**
- **Less HW dependency.**
- **Simpler backup environment.**
- **Security Improvement.**
- **Access to many environments per user.**



Where are we now?

- Working on a stable prototype for SAS developers:
 - Users account management not easy....
 - User Authentication via ESA cosmos account (the same as the XMM-Newton Archive!!)
 - Still working how to set-up accessibility to different tools are repositories.
 - Working on SAS Bitbucket migration.
 - Writing documentation for SAS developers.
- Accessible via:
 - Local Visual Studio Code (testing other IDEs)



SAS virtual development environments



```
void ErrorDispatcher::fatal(ErrorHandler::Code n)
{
    // copy the std::string and reset the pointer, otherwise the exit() will
    // call the destructor and see a non-empty errstr.
    std::string msg = _errstr.str();
    _errstr.str("");
    _handler->fatal(n,msg);
}

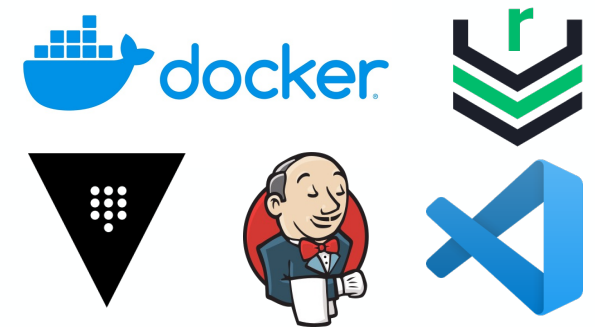
void ErrorDispatcher::error(ErrorHandler::Code n)
{
    std::string msg = _errstr.str();
    _errstr.str("");
    _handler->error(n,msg);
}

void ErrorDispatcher::warning(ErrorHandler::Code n)
{
    // Warning messages may be suppressed :
    if (_suppressor.suppress(n)) {
        _errstr.str("");
    } else {
        std::string msg = _errstr.str();
        _errstr.str("");
        _handler->warning(n,msg);
    }
}

void ErrorDispatcher::message(Layer layer, Verbosity level)
{
    std::string msg = _errstr.str();
    throw(UnexpressionableRegion);
}

g++ -Wl,-rpath,$ORIGIN/./lib -Wl,-rpath,$ORIGIN/./libextra -L/sasbuild/sasdev/lib -L/sasbuild/build/docker/GNU_CC_CXX_11.3.0/devtrack_dev_11.3.0_build/lib -o evselect evsele
ct_main.o -lvevselect -lmetatask -lcal -lselector -lcaloalutils -lselector -lslatec -ldal -lerror -lparam -lselector -lcaloalutils -ldal -lslatec -lutils -lselector -ldss -ldatata
ils -lmetatask -L/sasbuild/local/docker/GNU_CC_CXX_11.3.0/qt-x11-free/lib -lqt -lqfortran -lcfitsio -ldl -lm -L.
[sasbuild@docker evselect]$
```

SAS Virtual Development Environment



SAS Technology Stack

Next Steps...



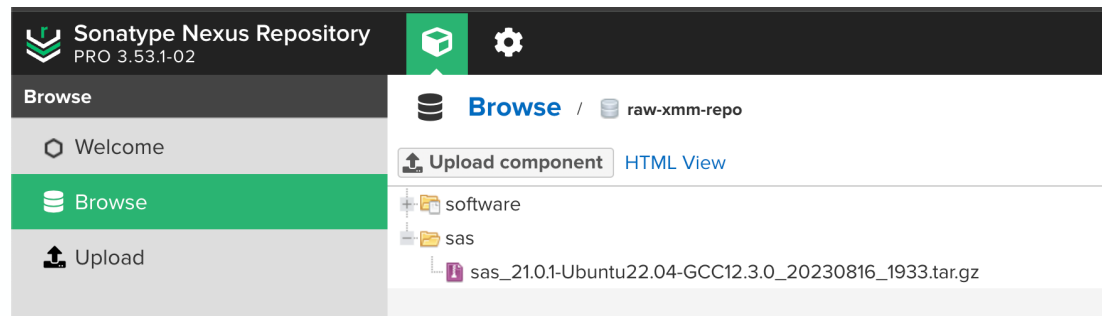
→ THE EUROPEAN SPACE AGENCY

SAS Virtual Development Environments



Steps to set-up SAS VDE

- 1.- Authenticate, Download and Run SAS Docker builder Image from ESA Nexus repository.
- 2.- Download or have access to CCF repository in your host machine.



```
#> docker pull scidockreg.esac.esa.int:61900/xmm/sasbuilder/ubuntu-22.04/gcc-11.3.0:1.0.0-3
```

```
#> mkdir /pathToCCF/ccf
```

```
#> rsync -v -a --delete --delete-after --force --include='*.CCF' --exclude='*/' sasdev-xmm.esac.esa.int::XMM_CCF /pathToCCF/
```



Steps to set-up SAS VDE

3.- Run de SAS builder image

4.- Build SAS

```
#> ./docker_run_it.sh -o ubuntu-22.04 -g gcc-11.3.0
```

```
#> sasbuilder/bin/buildsas "" "11.3.0" devtrack dev
```

```
aibarra@xmm1110:~/SAS_DOCKER/SASDEV$ ./docker_run_it.sh -o ubuntu-22.04 -g gcc-11.3.0
Current UID:GID : 5159:100
Using group users with gid 100
Generating locales (this might take a while)...
  en_US.UTF-8... done
Generation complete.
. /sasbuild/local/docker_ubuntu_22_04/GNU_CC_CXX_11.3.0
Workstation docker_ubun. /sasbuild/local/docker_ubuntu_22_04/GNU_CC_CXX_11.3.0/headas/architecture/headas-init.sh
Linux Ubuntu22.04 x86_64

[sasbuild@docker_ubuntu_22_04 ~]$

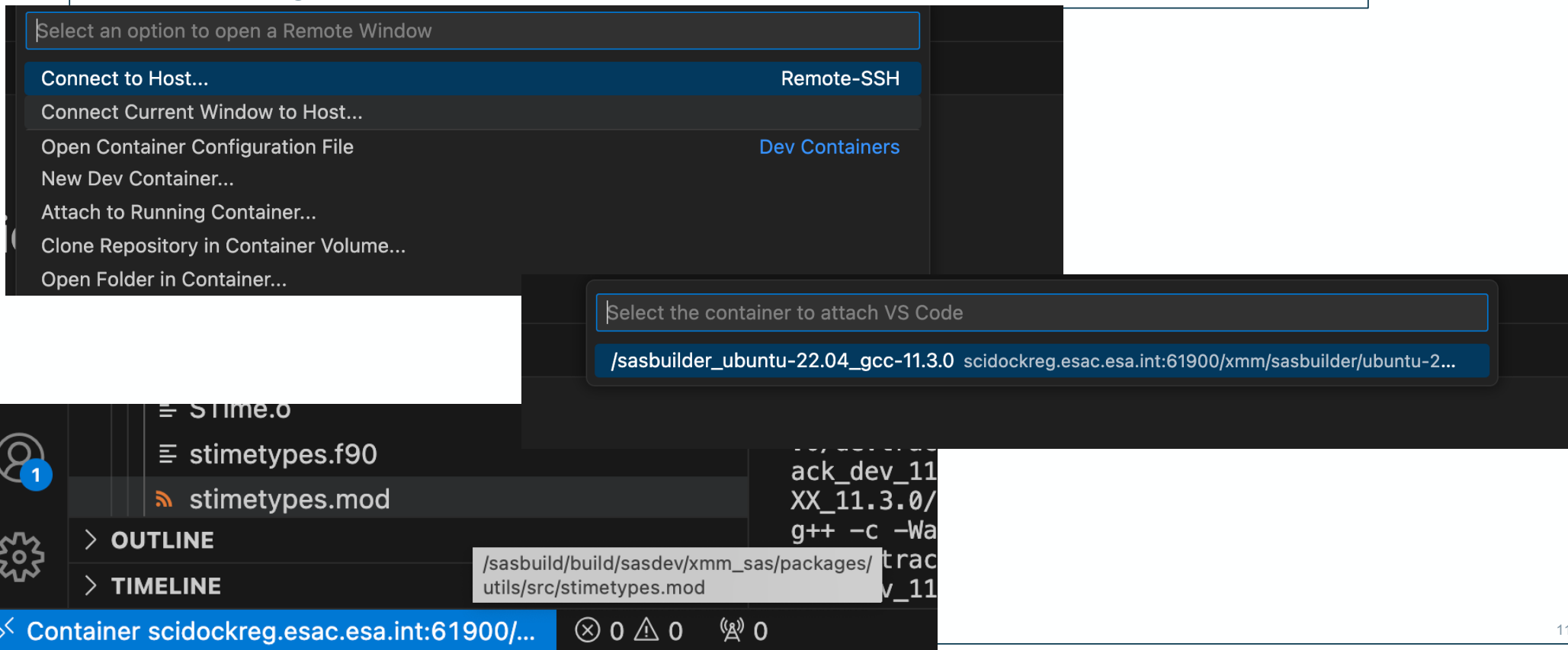
sasbuild@go-vm5f03f64e:~/docker_dev/wa04_sas_builder_docker$ ./docker_run_it.sh -o ubuntu-22.04 -g gcc-11.3.0
Current UID:GID : 5153:200
Generating locales (this might take a while)...
  en_US.UTF-8... done
Generation complete.
Workstation docker_ubuntu_22_04
Linux Ubuntu22.04 x86_64

[sasbuild@docker_ubuntu_22_04 ~]$ sasbuilder/bin/buildsas "" "11.3.0" devtrack dev
Warning: buildsas: Setting default builder configuration
```



Steps to set-up SAS VDE

5.- Start coding!!!

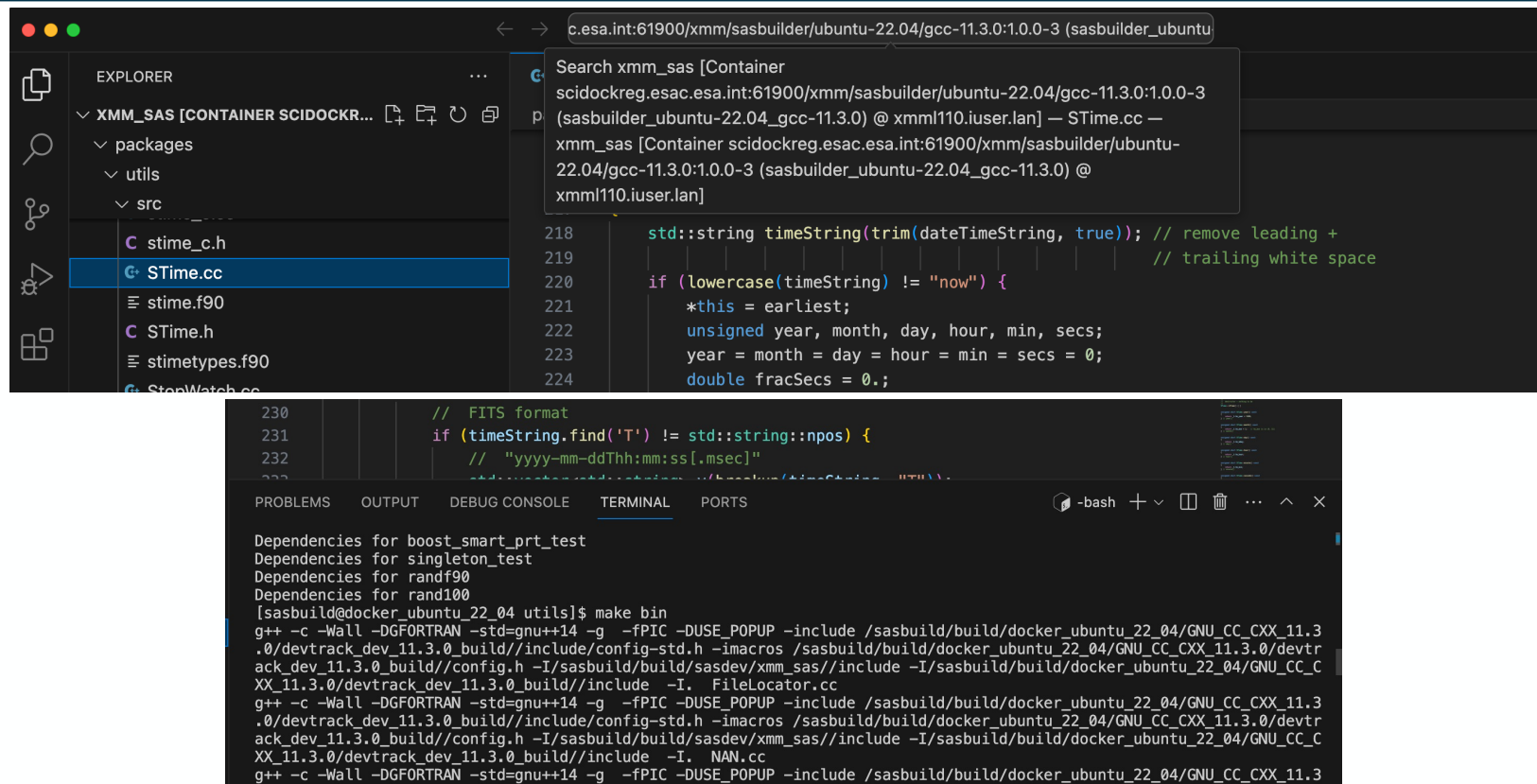


The screenshot displays the Visual Studio Code interface with several key elements:

- Remote Window Selection:** A dropdown menu titled "Select an option to open a Remote Window" is open, showing options like "Connect to Host... Remote-SSH" and "Open Container Configuration File Dev Containers".
- Container Selection:** A second dropdown menu titled "Select the container to attach VS Code" is open, showing a selected container: `/sasbuilder_ubuntu-22.04_gcc-11.3.0 scidockreg.esac.esa.int:61900/xmm/sasbuilder/ubuntu-2...`.
- File Explorer:** The left sidebar shows a file tree with folders like `stimetypes.f90` and `stimetypes.mod`.
- Terminal:** The bottom terminal window shows a command: `g++ -c -Wa` and a file path: `/sasbuild/build/sasdev/xmm_sas/packages/...`.
- Status Bar:** The bottom status bar indicates the active container: `<< Container scidockreg.esac.esa.int:61900/...`.

Steps to set-up SAS VDE

5.- Start coding!!!



The screenshot displays a code editor with a file explorer on the left showing a project structure with folders 'packages', 'utils', and 'src', and files 'stime_c.h', 'STime.cc', 'stime.f90', 'STime.h', 'stimetypes.f90', and 'StopWatch.cc'. The main editor shows C++ code for parsing a time string into FITS format. A search box is open over the code. Below the editor is a terminal window showing the output of a 'make bin' command, including dependency lists and compilation flags.

```
std::string timeString(trim(dateTimeString, true)); // remove leading +
// trailing white space
if (lowercase(timeString) != "now") {
    *this = earliest;
    unsigned year, month, day, hour, min, secs;
    year = month = day = hour = min = secs = 0;
    double fracSecs = 0.;
}

// FITS format
if (timeString.find('T') != std::string::npos) {
    // "yyyy-mm-ddThh:mm:ss[.msec]"
    std::vector<std::string> v(boost::split(timeString, "T"));
}

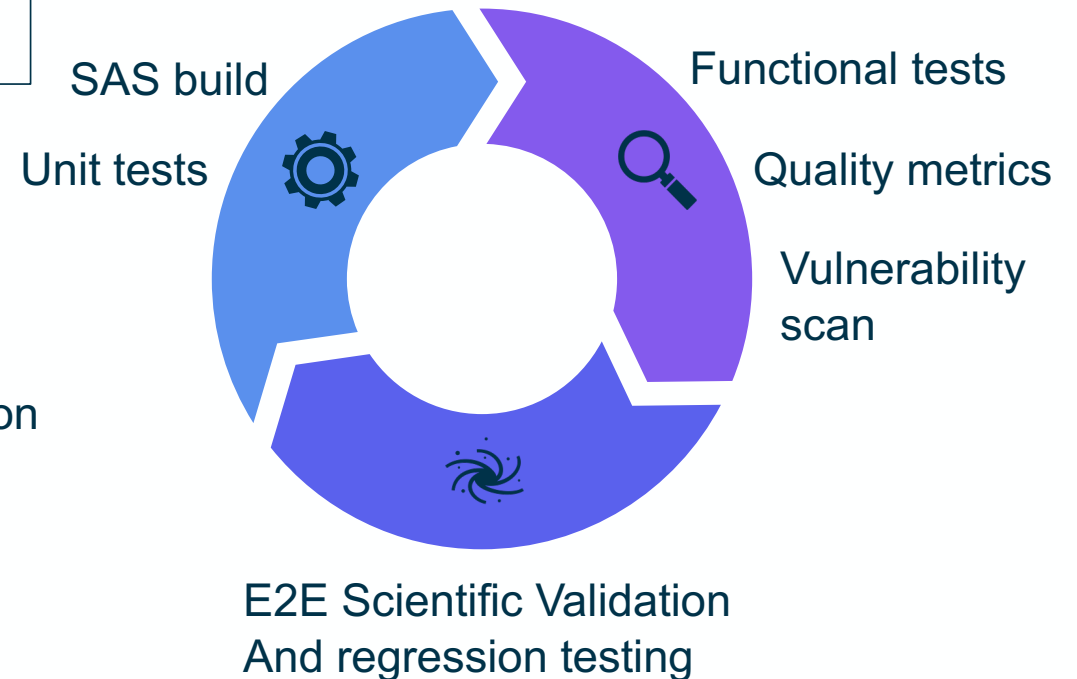
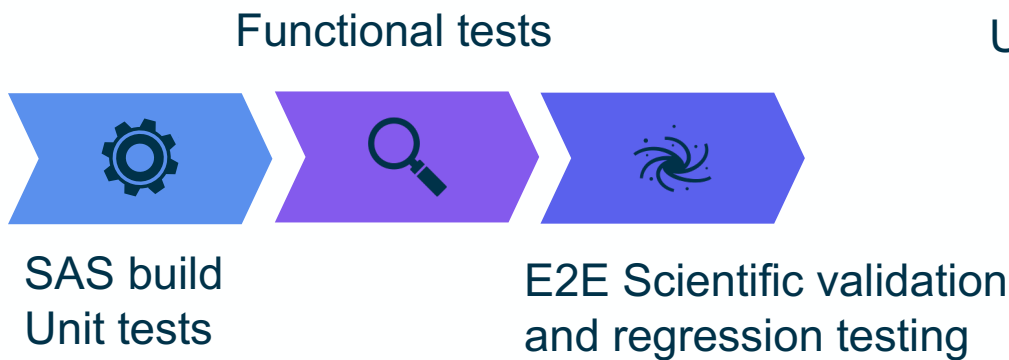
Dependencies for boost_smart_ptr_test
Dependencies for singleton_test
Dependencies for randf90
Dependencies for rand100
[sasbuild@docker_ubuntu_22_04 utils]$ make bin
g++ -c -Wall -DGFORTTRAN -std=gnu++14 -g -fPIC -DUSE_POPUP -include /sasbuild/build/docker_ubuntu_22_04/GNU_CC_CXX_11.3
./devtrack_dev_11.3.0_build/include/config-std.h -imacros /sasbuild/build/docker_ubuntu_22_04/GNU_CC_CXX_11.3.0/devtr
ack_dev_11.3.0_build/config.h -I/sasbuild/build/sasdev/xmm_sas//include -I/sasbuild/build/docker_ubuntu_22_04/GNU_CC_C
XX_11.3.0/devtrack_dev_11.3.0_build/include -I. FileLocator.cc
g++ -c -Wall -DGFORTTRAN -std=gnu++14 -g -fPIC -DUSE_POPUP -include /sasbuild/build/docker_ubuntu_22_04/GNU_CC_CXX_11.3
./devtrack_dev_11.3.0_build/include/config-std.h -imacros /sasbuild/build/docker_ubuntu_22_04/GNU_CC_CXX_11.3.0/devtr
ack_dev_11.3.0_build/config.h -I/sasbuild/build/sasdev/xmm_sas//include -I/sasbuild/build/docker_ubuntu_22_04/GNU_CC_C
XX_11.3.0/devtrack_dev_11.3.0_build/include -I. NAN.cc
g++ -c -Wall -DGFORTTRAN -std=gnu++14 -g -fPIC -DUSE_POPUP -include /sasbuild/build/docker_ubuntu_22_04/GNU_CC_CXX_11.3
./devtrack_dev_11.3.0_build/include/config-std.h -imacros /sasbuild/build/docker_ubuntu_22_04/GNU_CC_CXX_11.3.0/devtr
```



SAS building process based on dockers

Goal

- Avoid hardware dependencies
- To have a bit more agile Continuous Integration System
- Get ready to move the integration system to a cloud platform




SAS in Datalabs



→ THE EUROPEAN SPACE AGENCY

ESA Datalabs [0.7.0/BETA]

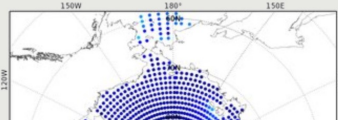


«YOU CAN EITHER MOVE YOUR QUESTIONS OR THE DATA. [...] OFTEN IT TURNS OUT TO BE MORE EFFICIENT TO MOVE THE QUESTIONS THAN TO MOVE THE DATA.»

Jim Gray, eScience: A Transformed Scientific Method

BRING YOUR QUESTIONS TO THE DATA

There is a new paradigm, opening completely new opportunities for discovery – a data-intensive approach to science. In many domains, we have entered what could be called the golden age of surveys, with several large-scale projects, spanning decades, between finished, ongoing, and planned activities. ESA is



- Registration is subject to moderation
- Personal and team workspace
- Data Volumes
- Pipelines
- Datalabs



SAS in Datalabs



→ THE EUROPEAN SPACE AGENCY esa

ESA Datalabs [0.7.0/BETA] 🧪 📁 🗂️ 📧 👤

Datalabs

Manage your running datalabs + Launch new

xmm
jl-xmm-sas

🔌 Delete

Data Volume Catalog

Domain
 Space Science (1)

🔍 ⚙️

XMM-Newton CCFs
Data Volume for XMM-Newton Calibration CCFs repository. Data volume made available by XMM-Newton mission.



SAS evolution along the years: DL4SAS + Threads



- To help users to analyse XMM-Newton data, a set of Data Analysis Threads are provided to the community.

SAS THREADS

JUPYTER NOTEBOOK THREADS

With the infrastructure of Python introduced in SAS 1.0, three experimental threads have been released under Jupyter Notebooks. These threads are not intended to be complete but to serve the purpose of illustrating how to use the Python interface to run SAS from a Jupyter Notebook.

SAS Start-up and event list manipulation		
- SAS start-up thread in Python	Jupyter Notebook	html
- How to reprocess ODFs to generate calibrated and concatenated EPIC event lists	Jupyter Notebook	html
- How to filter EPIC event lists for flaring particle background	Jupyter Notebook	html

COMMON THREADS

Starting the SAS

- SAS start-up	command line
----------------	--------------

All in one go: from raw data (ODF) to science products

- Analysis chain for point-like sources: <i>xmextractor</i>	command line
-------------------------------------------------------------	--------------

Guidelines for scientific analysis

- Spectral analysis with <i>XSPEC</i>	command line
- Timing analysis with <i>XRONOS</i>	command line

EPIC RELATED THREADS

All in one go: from raw data (ODF) to science products

- Analysis chain for point-like sources: <i>xmextractor</i>	command line
-------------------------------------------------------------	--------------

Step-by-Step

ESA Datalabs [0.5.0-43-G2FDDFF1C]

File Edit View Run Kernel Git Tabs Settings He

Filter files by name

/ my_workspace / SAS_Threads /

Name	Last Modified
epic-bkgfiltering_singleevt.ipynb	12 days ago
epic-reprocessing.ipynb	2 days ago
SAS_image_viewer.ipynb	2 days ago
sas-startup.ipynb	2 days ago
startsas.log	2 days ago

«YOU CAN EITHER MOVE YOUR QUESTIONS OR THE DATA. [...] OFTEN IT TURNS OUT TO BE MORE EFFICIENT TO MOVE THE QUESTIONS THAN TO MOVE THE DATA.»

BRING YOUR QUESTIONS TO THE DATA

There is a new paradigm, opening completely new opportunities for discovery – a data-intensive approach to science. In many domains, we have entered what could be called the golden age of surveys, with several large-scale projects, spanning decades, between finished, ongoing, and planned activities. ESA is responsible, or is a major partner, in several of these initiatives.

There is, however, a new profound change: data has become a major technological challenge. Increases by multiple orders of magnitude in dataset size means that transferring data to a scientist is often unfeasible.

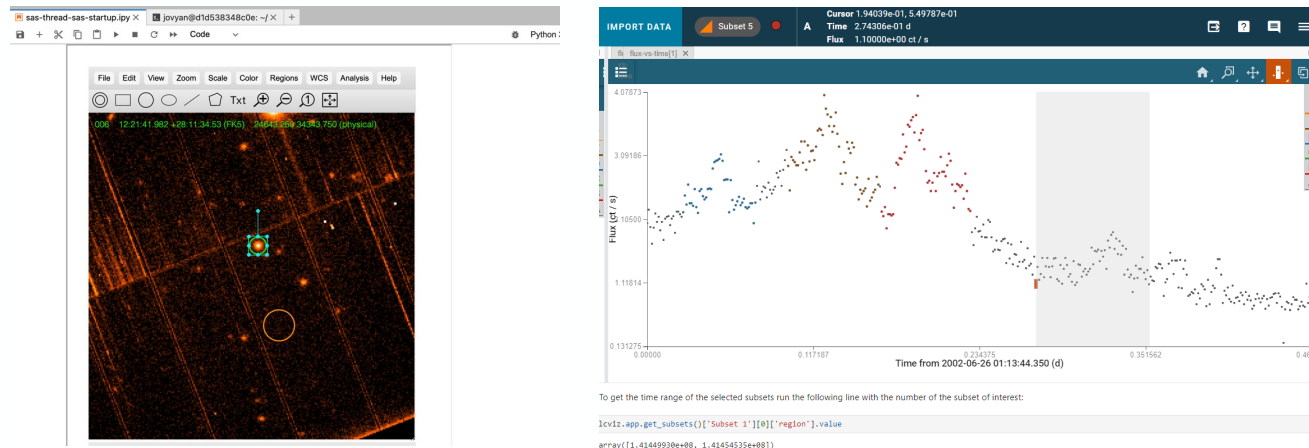
ESA datalabs gives you a privileged position; bring your code directly to ESA's infrastructure – there is a great set of tools and programming languages are flexible – and execute it with direct access to ESA's archives.



SAS on cloud infrastructures: Datalabs/SciServer??

- SAS is a docker can be used in any cloud platform.
- Scientific cloud platforms (Datalabs and SciServer) uses Jupyter Lab as user interface.
- SAS can be adapted to these platforms as long as we provide to use the basic functionalities to work with XMM-Newton data:
 - Image visualization
 - Light-curve visualization
 - Interactivity with these two functionalities
- Currently working on how to add interactivity in SAS Jupyter Lab environments using:

- jpyjs9
- Icviz



Future Work



- **Improve the SAS DevOps infrastructure**
 - **Fully automatic SAS building deployment indifferent cloud environments**
- **Improve/automatize SAS Virtual Development Environments**
- **Improve SAS & Datalabs (cloud) usage**
 - **Improve the SAS python infrastructure to help users to create their own scripts.**
 - **Interactivity**
 - **Source a background regions**
 - **Good Time Interval selection**
- **More things to come...**

