



Deep dive into the Xeus-based Cling kernel for Jupyter

# Sylvain Corlay



Founder and CEO of **QuantStack**

Open Source Developer

- **Jupyter Steering Committee Member**
- Core developer of **conda-forge**.
- Co-creator of Voilà, Xeus, Xtensor

Open Source volunteer work

- Director at **NumFOCUS**
- Organizer of the **PyData Paris** Meetup, vice chair of **JupyterCon**.

🏆 Recipient of the **2017 ACM Software System Award** for Project Jupyter



@SylvainCorlay

# QuantStack

Scientific Computing

## QuantStack is

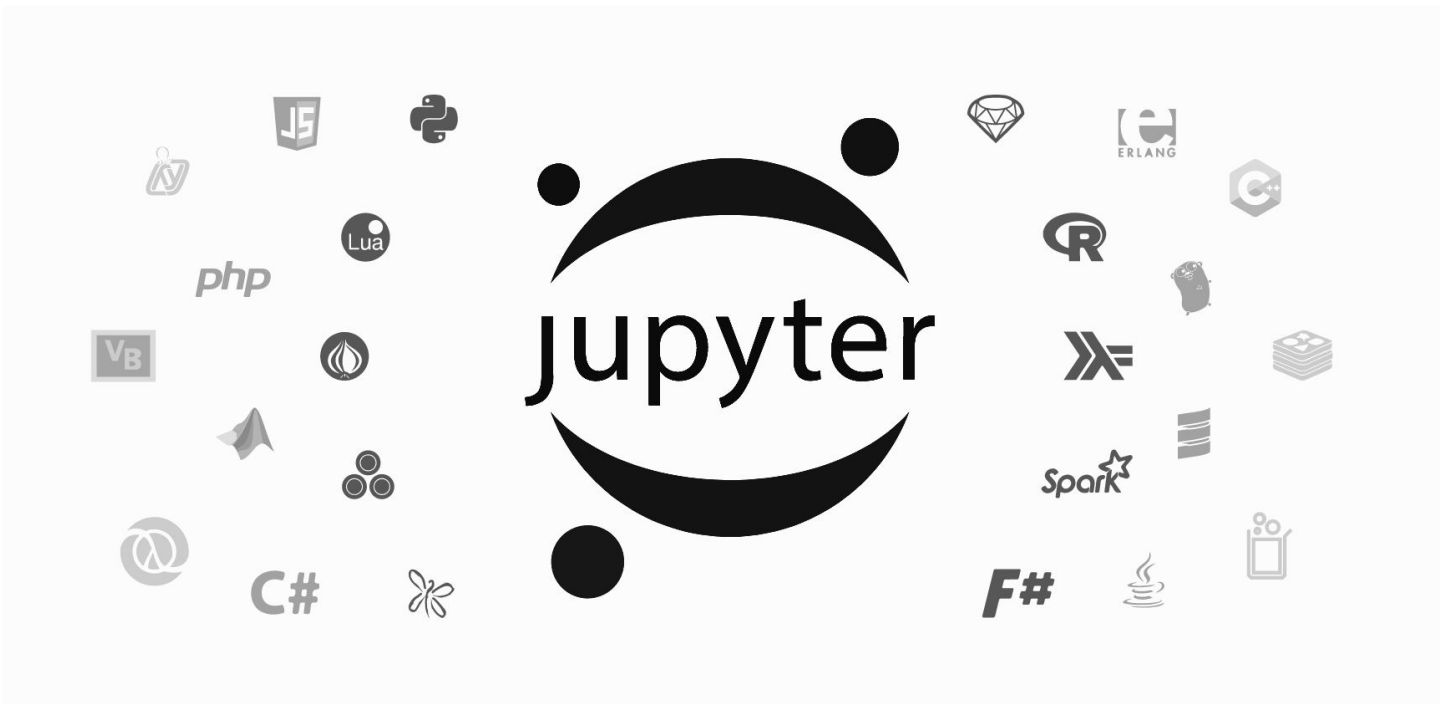
- An **open-source development studio** specialized in scientific computing
- A **team of maintainers of major opens-source projects** of the stack

(Jupyter, Conda-Forge, Xtensor, Voilà, Mamba, Quetz, ROS...)

## We provide

- professional support and development services for this ecosystem
- custom development and consulting services for the key software of the open-source scientific computing ecosystem.

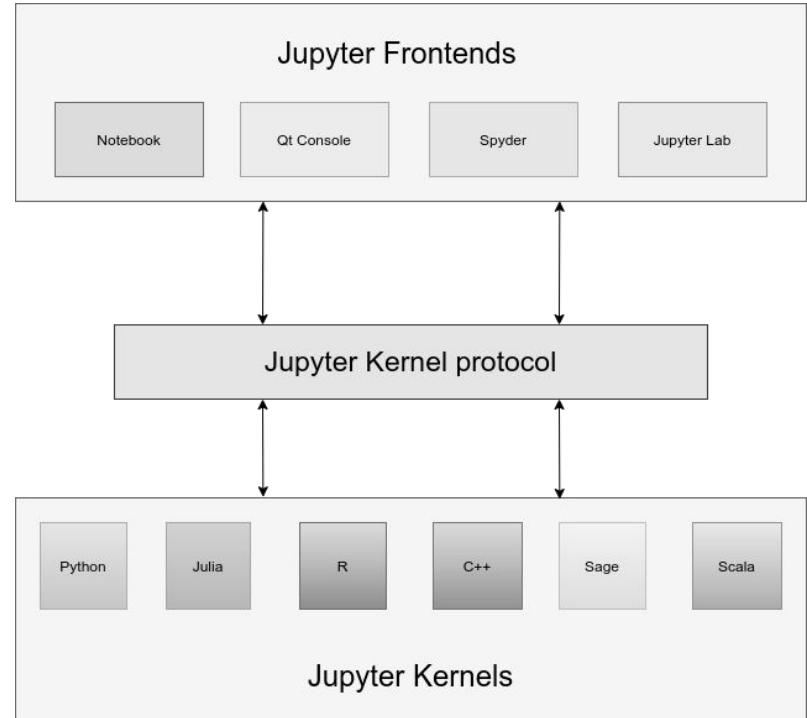
# Jupyter's language agnosticism



# Jupyter's language agnosticism

The **Kernel** is the part of the Jupyter infrastructure responsible for executing the user's code.

From the perspective of the other components of the Jupyter stack, a kernel is merely a process implementing a **well-specified communication protocol**.



# Authoring Jupyter kernels

## The existing kernels:

JavaScript, C++, Python, Julia, R, Haskell, Go, C#, Robotframework, OCaml, Perl, IDL, Scala, Fortran, Octave, Scilab, SQLite, Ruby...

**There are Jupyter kernels for dozens of languages.**

But these kernels have very different levels of quality and support for the features of the protocol.

## How to make new language kernel?

1. Rewrite in **from scratch** in e.g. the target language... Not that easy.
  - Deal with a complex concurrent programming models
  - Make use of the ZMQ interprocess communication library
  - Cryptographically sign messages
  - Properly implement JSON messages schemas
2. Use a **framework**
  - Ipykernel
  - Xeus

# Authoring Jupyter kernels

## The wrapper kernel approach

IPykernel includes a reference implementation of the Kernel protocol.

To make a kernel with ipykernel, inherit from `ipykernel.kernelbase.Kernel` and implement the language-specific parts in the derived class.

This is the approach used for the kernel shipped with Cling.

## Issues with the wrapper approach:

- Dependency on the Python runtime. *(consequences for the packaging of the Cling project).*
- The wrapped interpreter may not have a Python API, and we need to make one.
- We may need to expose the API of the kernel to the target language for advanced use cases (widgets, rich display...).
- A native implementation may be more efficient.

# Authoring Jupyter kernels

## What is Xeus?

Xeus is a modern C++ implementation of the Jupyter protocol. It is *not* a kernel, but a *tool* to make new kernels.

To make a kernel with Xeus, inherit from **`xeus::xinterpreter`** and implement the language-specific parts in the derived class.

This is the approach used in `xeus-cling`.

## Our motivation for starting Xeus

- We were asked by a client to make a lightweight kernel for a DSL. IPykernel seemed overkill and too heavy.
- We think that the kernel protocol is stable enough for a strongly typed reference implementation to exist.
- Most interpreters are written in C or offer a C API. This makes it easy to embed them in a C++ application.



# Xeus: an ecosystem of Jupyter kernels

**Xeus-python:** A xeus-based Jupyter kernel for the Python language

- [GitHub](#) [Try it Here](#)
- Used in SlicerJupyter for embedding in the Slicer Qt application.
- Supports the new JupyterLab interactive debugger.

**Xeus-cling:** A xeus & cling-based Jupyter for the C++ language

- [GitHub](#) [Try it Here](#)
- Started as a demonstrator for the Xeus framework. Used to teach C++ at Université Paris Sud.

**Xeus-SQL:** (And Xeus-SQLite): Xeus-based kernels for SQL

- [GitHub](#) [Try it Here](#)

**Xeus-Robot:** Xeus-based kernel for RobotFramework

- [GitHub](#) [Try it Here](#)
- RobotFramework is an open-source language and framework for Robotic Process Automation.

**LFortran:** LFortran is an LLVM-based Fortran compiler and interpreter. It includes a Xeus-based kernel

- [GitHub](#) [Try it Here](#)

```
$ jupyter console --kernel=fortran
Run with XEUS 0.24.1
Jupyter console 6.1.0

LFortran
Jupyter kernel for Fortran
Fortran
In [1]: integer :: x

In [2]: x = 5

In [3]: x*3.5
Out[3]: 17.500000

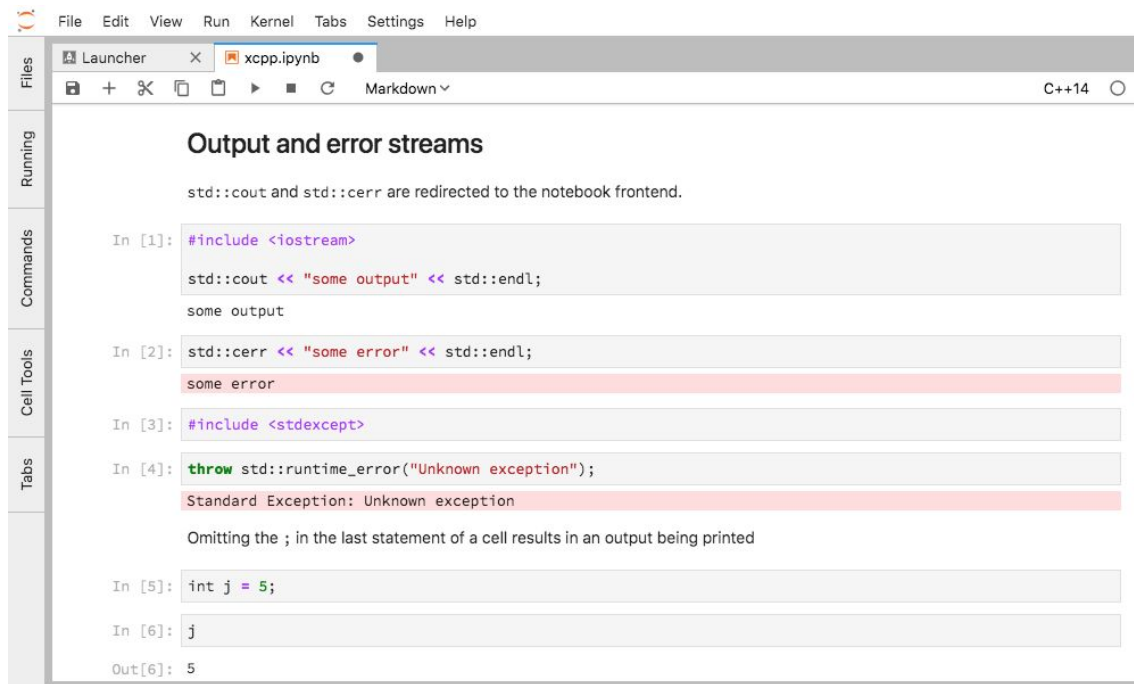
In [4]: █
```

**And many more (xeus-octave,  
xeus-fift, JuniperKernel)...**

# Xeus-cling: A C++ Jupyter kernel ... based on cling and Xeus

*Never give a live demo*

# Xeus-cling: redirecting streams



The screenshot shows a Jupyter notebook titled "xcpp.ipynb" in the Xeus-cling environment. The notebook content is as follows:

```
Output and error streams

std::cout and std::cerr are redirected to the notebook frontend.

In [1]: #include <iostream>

std::cout << "some output" << std::endl;
some output

In [2]: std::cerr << "some error" << std::endl;
some error

In [3]: #include <stdexcept>

In [4]: throw std::runtime_error("Unknown exception");
Standard Exception: Unknown exception

Omitting the ; in the last statement of a cell results in an output being printed

In [5]: int j = 5;

In [6]: j

Out[6]: 5
```

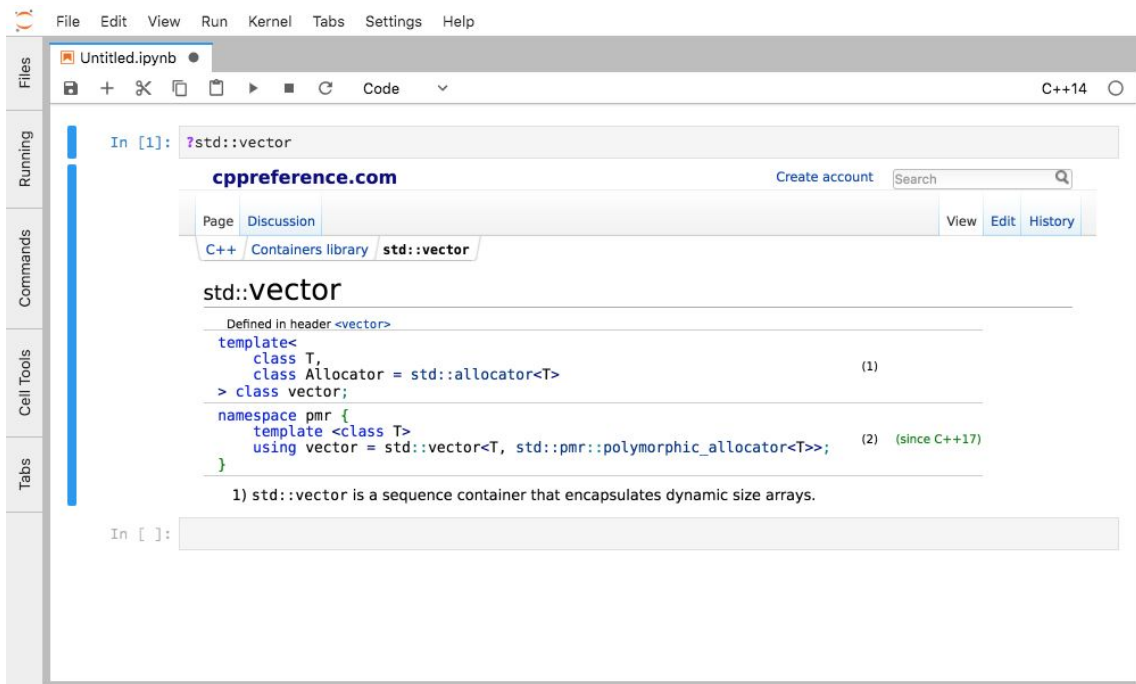
**The main means of printing are redirected to the front-end.**

- **std::cout** and **std::cerr**, as well as **printf** are redirected to the front-end.

However.

- **std::clog** prints to the kernel standard output, which can be used for logging.

# Xeus-cling: inline help



The screenshot shows the Xeus-cling interface with a terminal window displaying the command `?std::vector`. The output shows the Cppreference.com website for `std::vector`. The code block contains the following content:

```
In [1]: ?std::vector

cppreference.com Create account Search
Page Discussion View Edit History
C++ Containers library std::vector

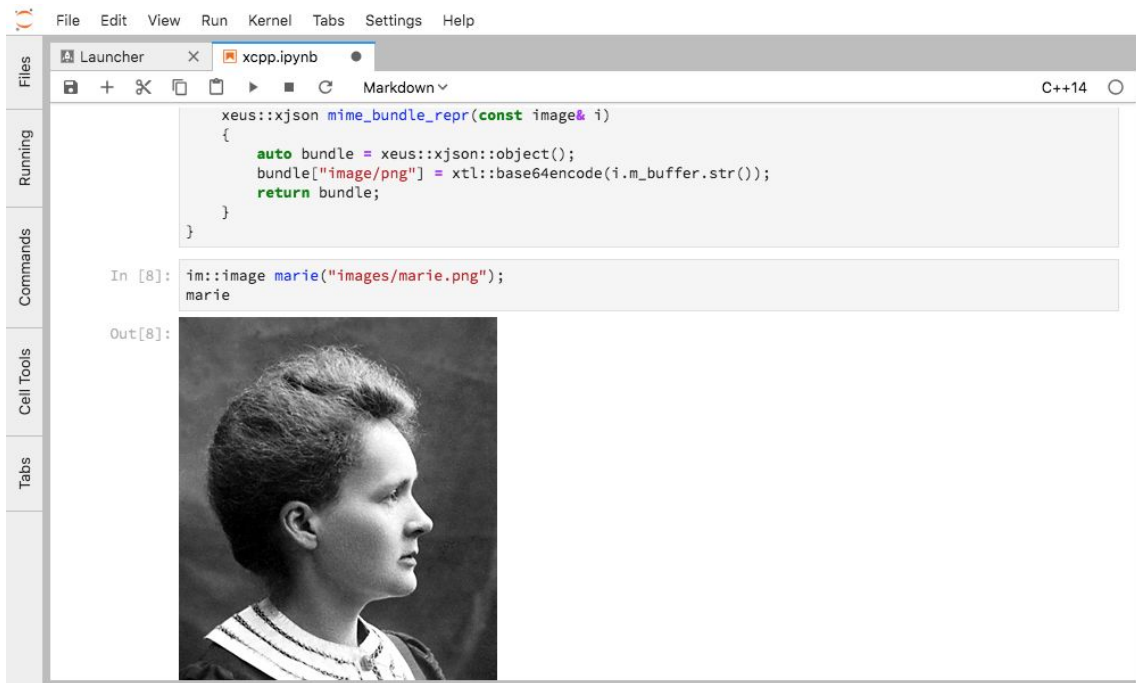
std::vector
Defined in header <vector>
template<
  class T,
  class Allocator = std::allocator<T>
> class vector;
namespace pmr {
  template <class T>
  using vector = std::vector<T, std::pmr::polymorphic_allocator<T>>;
}

1) std::vector is a sequence container that encapsulates dynamic size arrays.
```

**The "?" magic can be used to get inline help on types and functions.**

- For the standard library makes use of `cppreference`.
- This is extensible for user-defined libraries. (Demo example with `xtensor`)

# Xeus-cling: rich outputs




The screenshot displays the Xeus-cling Jupyter interface. The top menu bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The interface shows a code cell with the following C++ code:

```
xeus::xjson mime_bundle_repr(const image& i)
{
    auto bundle = xeus::xjson::object();
    bundle["image/png"] = xtl::base64encode(i.m_buffer.str());
    return bundle;
}
```

Below the code cell, the input and output are shown:

```
In [8]: im::image marie("images/marie.png");
marie
```

Out[8]:



**Xeus-cling leverages the Jupyter rich mime type rendering system.**

- This can be defined for any type by specializing the **mime\_bundle\_repr** function for the said type.
- This overload is picked up by xeus-cling through **argument dependent lookup**.

# Rich output

## Examples with

- **Xtensor** and **Xframe** (HTML tables for visualizing tensors)
- **Symengine** (MathJax)

Installation

Install `xeus-cling`:

```
conda install -c conda-forge xeus-cling
```

```
[1]: #include <symengine/expression.h>
using SymEngine::Expression;
```

```
[2]: Expression x("x");
```

```
[3]: auto ex = pow(x+sqrt(Expression(2)), 10);
ex
```

```
[3]:  $(x + \sqrt{2})^{10}$ 
```

```
[4]: expand(ex)
```

```
[4]:  $32 + 160\sqrt{2}x + 960\sqrt{2}x^3 + 1008\sqrt{2}x^5 + 240\sqrt{2}x^7 + 10\sqrt{2}x^9 + 720x^2 + 1680x^4 + 840x^6 + 90x^8 + x^{10}$ 
```

Using `linspace`, `arange`, `ones`, `zeros`

```
[11]: #include "xtensor/xbuilder.hpp"
```

```
[12]: xt::xarray<double> ar = xt::linspace<double>(0.0, 10.0, 12);
ar.reshape({4, 3});
ar
```

0.	0.909091	1.818182
2.727273	3.636364	4.545455
5.454545	6.363636	7.272727
8.181818	9.090909	10.

```
[13]: xt::xarray<double> fones = xt::ones<float>({2, 2});
fones
```

1.	1.
1.	1.

```
[5]: auto time_axis = xf::axis({"2018-01-01", "2018-01-02", "2018-01-03", "2018-01-04", "2018-01-05", "2018-01-06"});
```

```
[6]: auto dry_temperature = variable_type(
dry_temperature_data,
{
{"date", time_axis},
{"city", xf::axis({"London", "Paris", "Brussels"})}
});
```

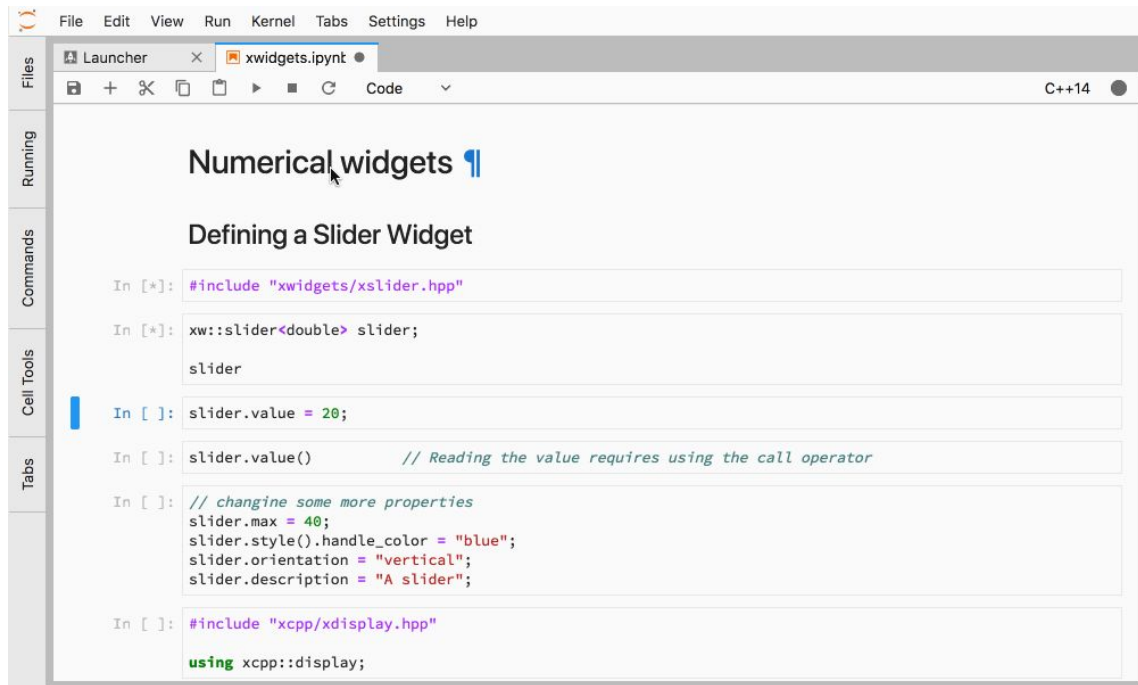
```
[7]: dry_temperature
```

	London	Paris	Brussels
2018-01-01	N/A	23.3501	24.6887
2018-01-02	17.2103	18.0817	20.4722
2018-01-03	16.8838	N/A	24.9646
2018-01-04	24.6769	22.2584	24.8111
2018-01-05	16.0986	22.9811	17.9783
2018-01-06	15.0478	16.1246	21.3976

## 1.2. Indexing and selecting data

Like `xarray`, `Xframe` supports four different kinds of indexing as described below:

# Xeus-cling: interactive widgets



The screenshot shows the Xeus-cling interface with a Jupyter notebook. The notebook has a title bar with 'Launcher' and 'xwidgets.ipynit'. The main content area displays the following code in a cell:

```
In [*]: #include "xwidgets/xslider.hpp"

In [*]: xw::slider<double> slider;
slider

In [ ]: slider.value = 20;

In [ ]: slider.value() // Reading the value requires using the call operator

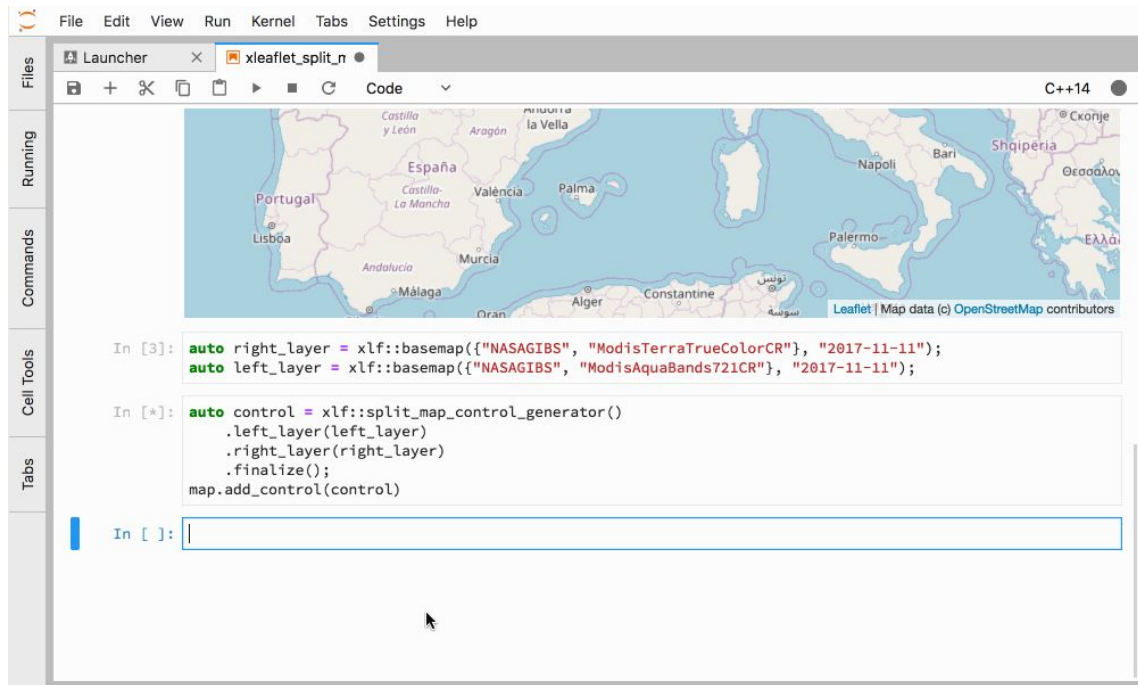
In [ ]: // change some more properties
slider.max = 40;
slider.style().handle_color = "blue";
slider.orientation = "vertical";
slider.description = "A slider";

In [ ]: #include "xcpp/xdisplay.hpp"
using xcpp::display;
```

## Interactive widgets

- A C++ backend for the Jupyter interactive widgets is available in the xwidgets package.

# Xeus-cling: more data visualization



The screenshot displays the Xeus-cling web interface. At the top, there is a menu bar with options: File, Edit, View, Run, Kernel, Tabs, Settings, Help. Below the menu is a browser-like tab bar with a tab titled 'xleaflet\_split\_n'. The main content area is divided into two sections. The upper section shows a map of the Mediterranean region, including parts of Spain, Portugal, Italy, and Greece. The map is overlaid with two data layers: 'NASAGIBS' (true color) and 'ModisAquaBands721CR' (water bands). The lower section is a code editor with the following code:

```
In [3]: auto right_layer = xlf::basemap({"NASAGIBS", "ModisTerraTrueColorCR"}, "2017-11-11");
auto left_layer = xlf::basemap({"NASAGIBS", "ModisAquaBands721CR"}, "2017-11-11");

In [*]: auto control = xlf::split_map_control_generator()
        .left_layer(left_layer)
        .right_layer(right_layer)
        .finalize();
        map.add_control(control);

In [ ]: |
```

## Jupyter widgets are a framework

- Xleaflet
- Xwebrtc
- Xplot

And many more coming...

An opportunity for interactive C++: leverage the huge ecosystem of JavaScript data visualization tools.



# Xeus Cling: how to get started

We provide a xeus-cling package on conda-forge.  
It can be installed with mamba or conda

```
mamba install xeus-cling
```

You can also try it out online on binder.

# Xeus Cling: about the future?

- Provide a VS2019 build on conda-forge to fully support windows
  - Windows support is tested on CI but we don't provide a build for it
  - We will wait for the LLVM9-based version of cling.
- Work with library authors on including cling pragmas in library headers
  - [https://github.com/xtensor-stack/xtensor-blas/blob/master/include/xtensor-blas/xblas\\_config\\_cling.hpp.in](https://github.com/xtensor-stack/xtensor-blas/blob/master/include/xtensor-blas/xblas_config_cling.hpp.in)
- Dashboarding with Voilà and Xeus-cling
  - build notebooks into full executables that don't require the cling runtime, and respond to the protocol as static backend for Voilà apps
  - Subject of an internship?
- Work with upstream on improving rich mime type rendering?
- What is needed for an upstream adoption in ROOT?
  - Provide an extensible magics system providing all the dots commands?
- Implementing the Jupyter Debug Protocol in xeus-cling to enable visual debugging in JupyterLab.