

Creating Apps and Deployment options

Marcus Åberg 2022-10-19



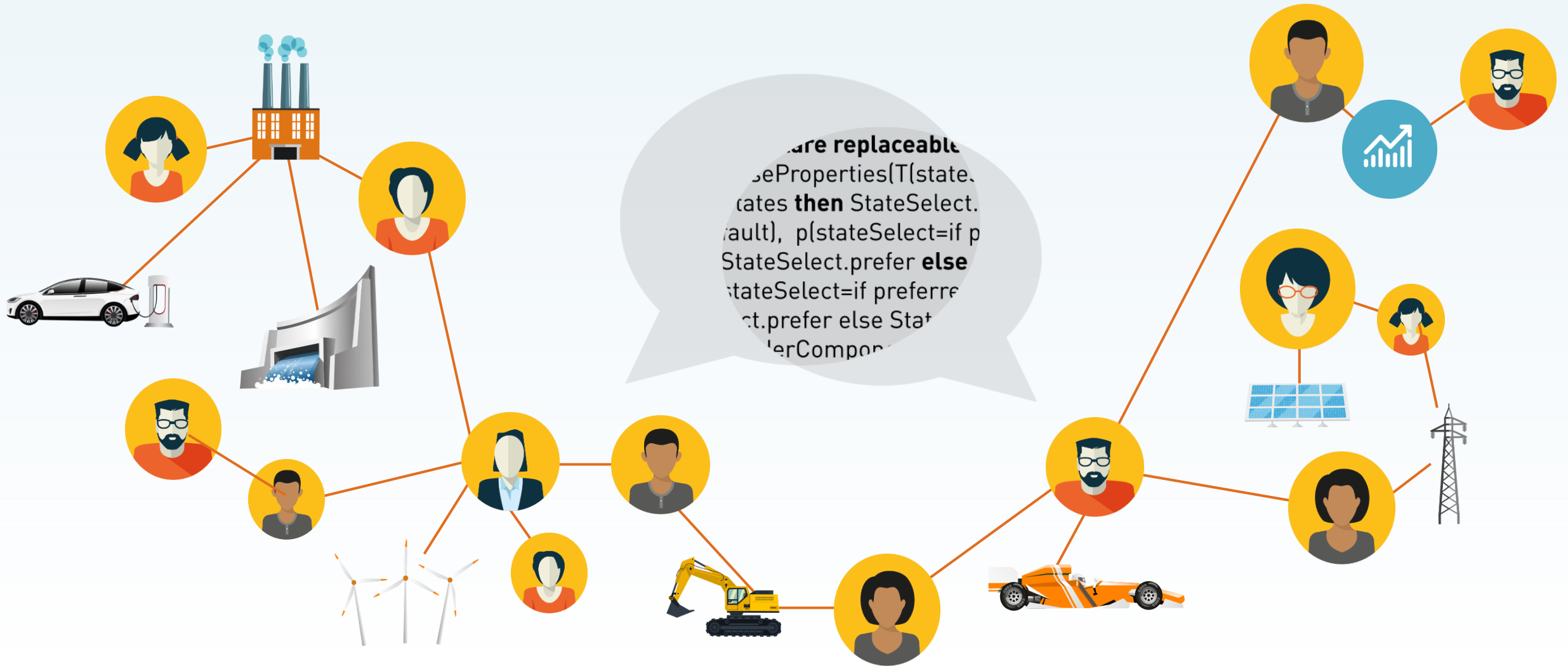
Outline

- Introduction
 - Why custom interfaces?
- Introducing - App Mode
 - Create a simplified interface for your model – fast!
- Leverage the rich Modelon Impact API for full freedom
 - Python Dashboarding
 - Traditional Web-Applications

Introduction

Why custom interfaces?

Few simulation specialists can make their know-how available...



...to enhance the capability of **many** engineers!

LESS \$ MORE



MODEL CREATION

= Investment cost

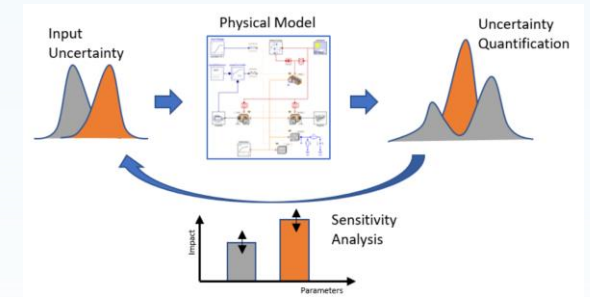
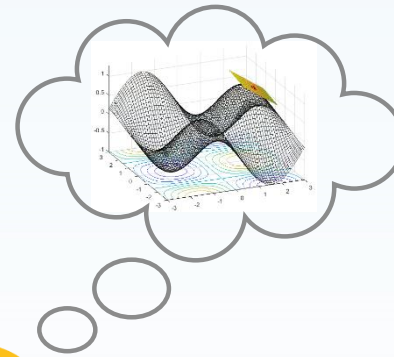
state replacement
useProperties(T(state, p, default), p(stateSelect=if preferredM
stateSelect=if preferredM
ct.prefer else StateSe
OrderComponent



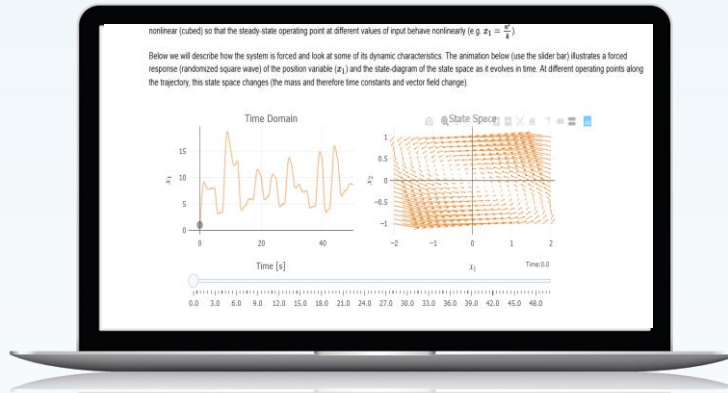
MODEL UTILIZATION

= Engineering Value

What is the optimal size of my component? What is my optimal system configuration?
What is the impact of manufacturing variability on system performance?



TIME TO GET PRODUCTIVE



are replace
 useProperties(T(state
 States then StateSelect.
 fault), p(stateSelect=if pre
 StateSelect.prefer else St
 stateSelect=if preferredM
 ct.prefer else StateSe
 OrderComponent



years

months

days

minutes

seconds



Many reasons – Same concept

- Power-user enablers
 - Streamline specialized pre-or post processing tasks
 - Reports
 - Model Calibration
 - Model Verification
- Deployment workflows
 - Large audience that could benefit from information provided by model
 - Custom interfaces enables these users to fetch this information directly

Different users benefit from different interfaces, tailored to their needs!

Introducing - App Mode

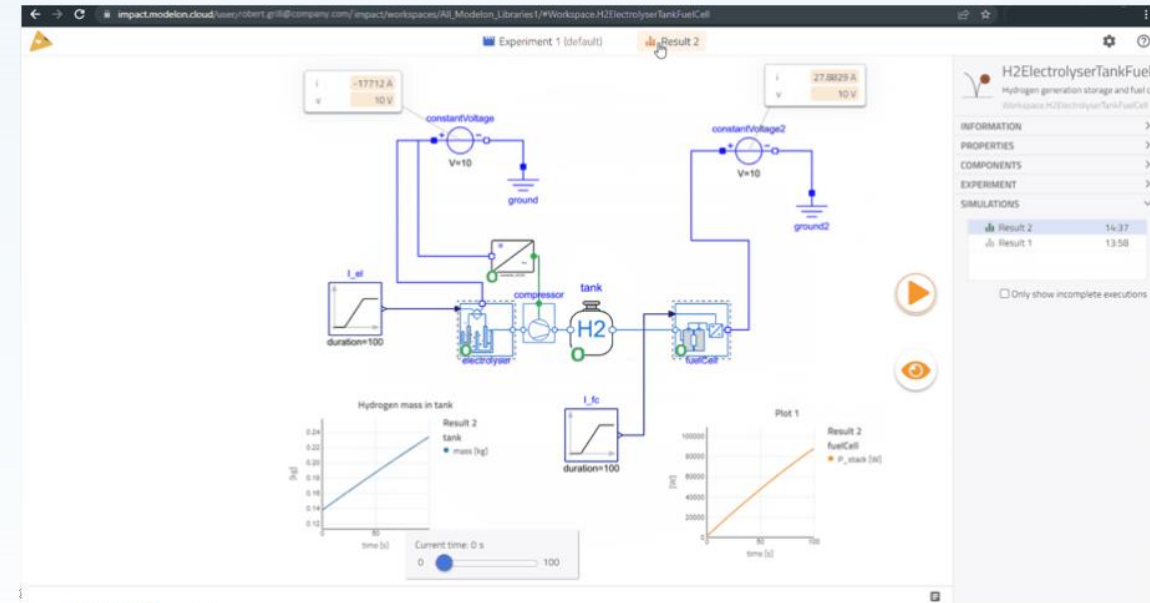
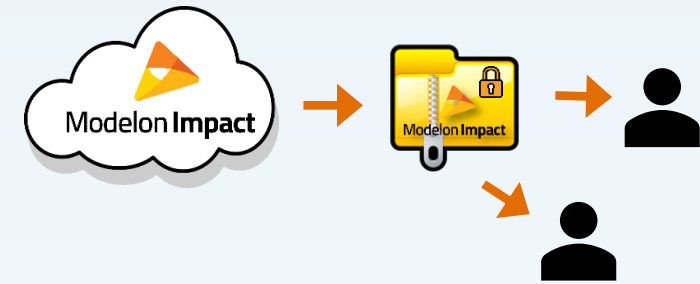
Create a simplified interface for your model – fast!

App Mode

Create a simplified interface to a model super fast!

Leveraging existing features in the MI UI:

- User interface prepared by Model author
- “Locked” mode, no editing allowed
- Interaction through stickies and views
- Portable - Share with others



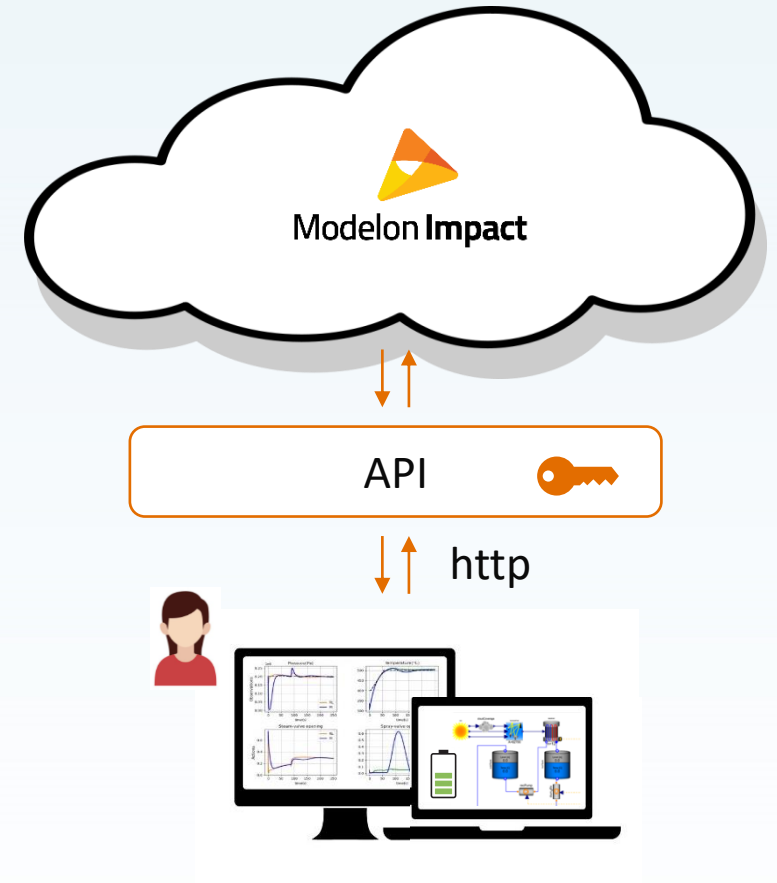
Demo – App Mode

The Modelon Impact API

Leverage Modelon Impact's rich API for full freedom

Modelon Impact (REST) API

- Modelon Impact's (REST) API enables communication with the Impact server over the Internet
- Possible to build customized interfaces on top of the API
- Implementation with a wide variety of programming languages and frameworks



https://help.modelon.com/latest/guides/rest_api/

API Overview

The API let's the developer:

- Authenticate against Modelon Impact
- Access resources, such as workspaces, models and results
- Define and execute experiments on the Impact server
- Download and access experiment results
- Upload model libraries

<https://help.modelon.com/latest/guides/apidocumentation.html>

Workspace Context Based

The concepts used in the API resembles normal user interaction.

Most operations in the API happens in the context of a Workspace:

- Models and dependencies
- Results
- Experiments

Everything is stored in the Workspace – can be investigated from the GUI

How to leverage the API

In Theory:

Any environment can connect that can:

- Connect to the Impact Server
- Send receive and interpret the HTTP requests/responses

In Practice:

Specific competence required to interact with the API directly:

- REST-API programming
 - HTTP-requests/response anatomy
- Efficient debugging
- Reading REST-API documentation

Introducing: Wrapper Libraries

Solution:

Simplified Interaction through Wrapper libraries:

- Python – general purpose/scripting
- JavaScript – for building web apps

Other benefits:

- Version handling

Python Wrapper

- Pre-Installed in Modelon Impact-Cloud
 - Open-source, can be installed with pip
- Can be used for:
 - Scripting, automating workflows
 - Creating interactive documentation and reports
 - Leveraging other packages and tools available in Python (for DoE, ML, Optimization etc.)
 - Using dashboard frameworks such as Dash, Voila, Steamlit etc.

JS-wrapper

- Leverage to create web applications
 - Open-source, installed with npm
- Web apps allow:
 - Simplified or Specialized interaction with models
 - Custom post-processing (plots, reports etc.)
 - Use of external web resources, APIs or services (maps ,weather etc.)

Microgrid Dashboard

Producers	Consumers	Economy
PV [MWh]	Load [MWh]	Cost: EL [\$]
DieselGen [MWh]		Cost: Diesel [\$]
Grid [MWh]		Fuel [l]

Python Dashboarding

Streamline your workflows and automate things with Python

Python Dashboarding Tools

Basically, what a python dashboarding tool does:

- Type some python code
- Show (interactive) visualization data
- Accessible through a browser

Some Benefits:

- Simplifies communication of results
- End user interacts and get the data they need

Modelon Impact Cloud includes: [Voila](#), [Dash](#)



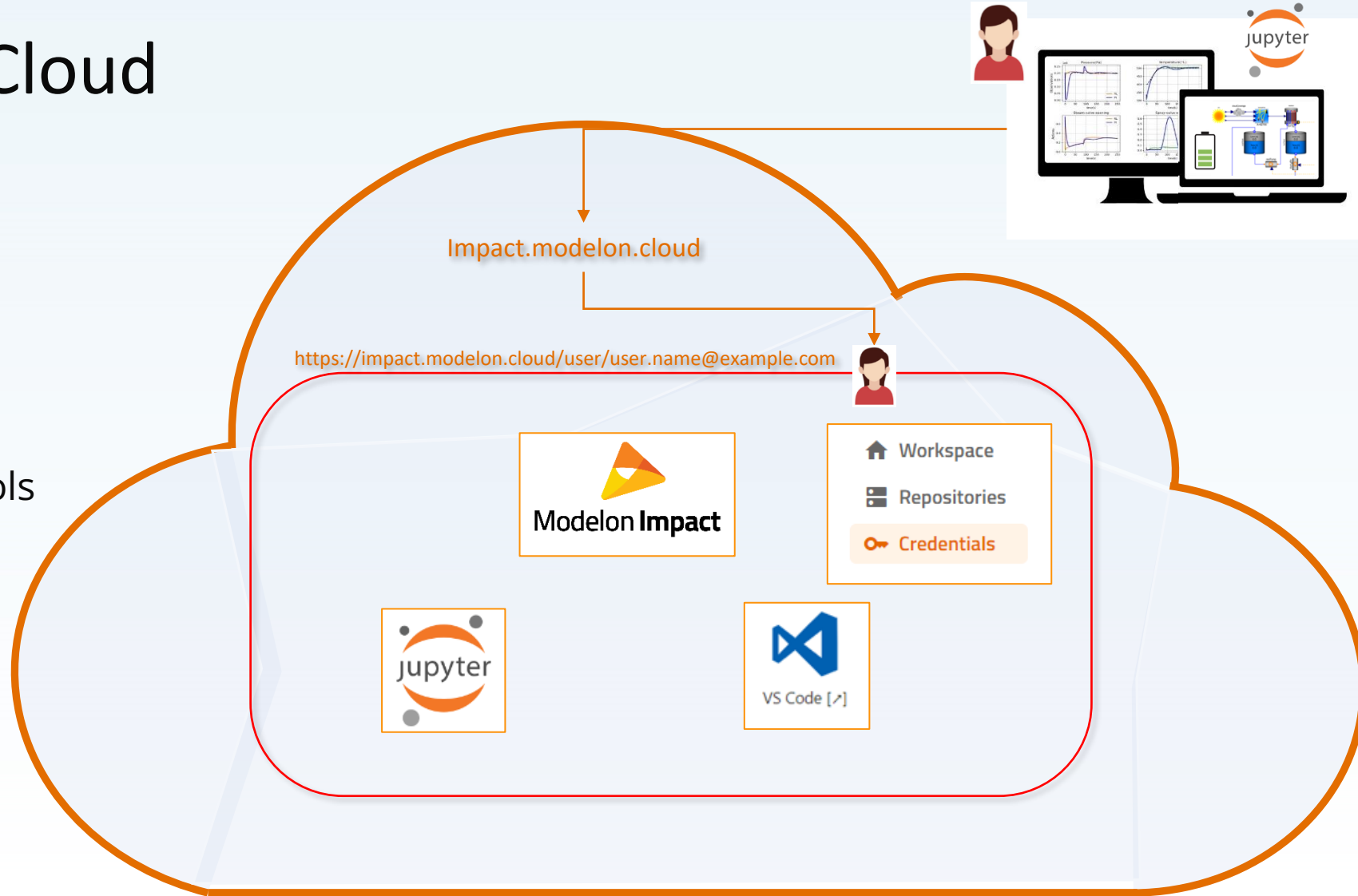
Modelon Impact Cloud

Apps available in the user profile:

- JupyterLab
- VSCode
- Workspace management Tools

Create custom apps using dashboarding tools.

Ideal for creating and automating custom workflows!



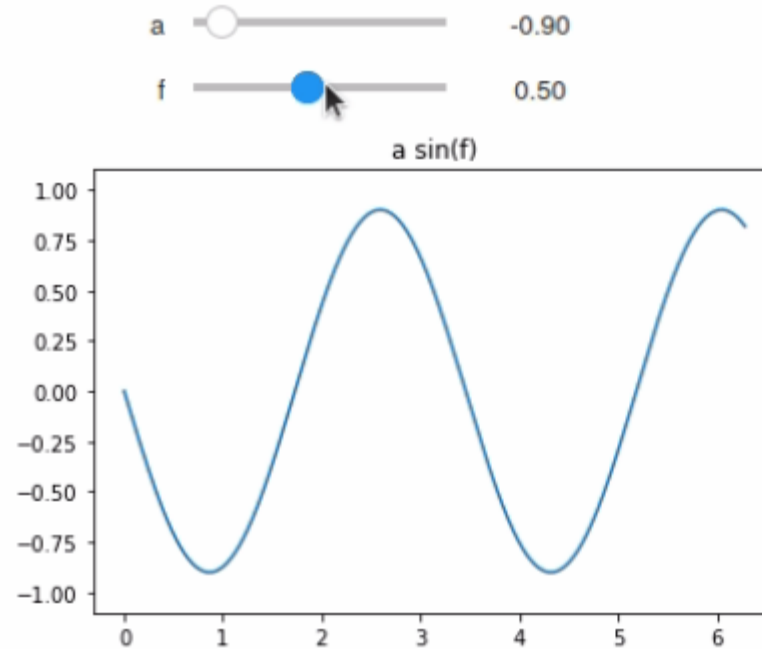
Example: Voila app

Voilà Framework:

- Converts a Jupyter Notebook to a standalone app
- Removes all code
- Text, Plots and Widgets constitutes UI

Voila Web App

A website built out of a Jupyter notebook using Voila



Example: Voila app

Develop:

- Create a Jupyter Notebook with your custom scripted workflow
- Add widgets and plots for user-interaction

```
[1]: import os
from urllib.parse import parse_qs

query_string = os.environ.get('QUERY_STRING', '')
parameters = parse_qs(query_string)

modelname = parameters.get('model', ['Sankey.Examples.Dummy'])[0]
workspaceid = parameters.get('workspaceid', ['Sankey'])[0]

print(f'Model to Simulate: {modelname}')
print(f'From workspace: {workspaceid}')

[2]: from modelon.impact.client import Client

client = Client('http://localhost:8888')
workspace = client.get_workspace(workspaceid)
dynamic = workspace.get_custom_function('dynamic')
model = workspace.get_model(modelname)

compiler_options = dynamic.get_compiler_options()
model_executable = model.compile(compiler_options=compiler_options).wait()

fmu_path = model_executable.download()

WARNING:modelon.impact.client.client:No API key could be found, will log in as anonymous user. Permissions may be limited
INFO:modelon.impact.client.operations:Cached FMU found! Using the cached FMU!

[3]: from pyfmi import load_fmu
import ipywidgets as widgets
import plotly.graph_objects as go
from IPython.display import display

w_starttime = widgets.FloatText(
    value=0,
    description="Start time:",
    disabled=False
)

w_finaltime = widgets.FloatText(
    value=1,
    description="Final time:",
    disabled=False
)

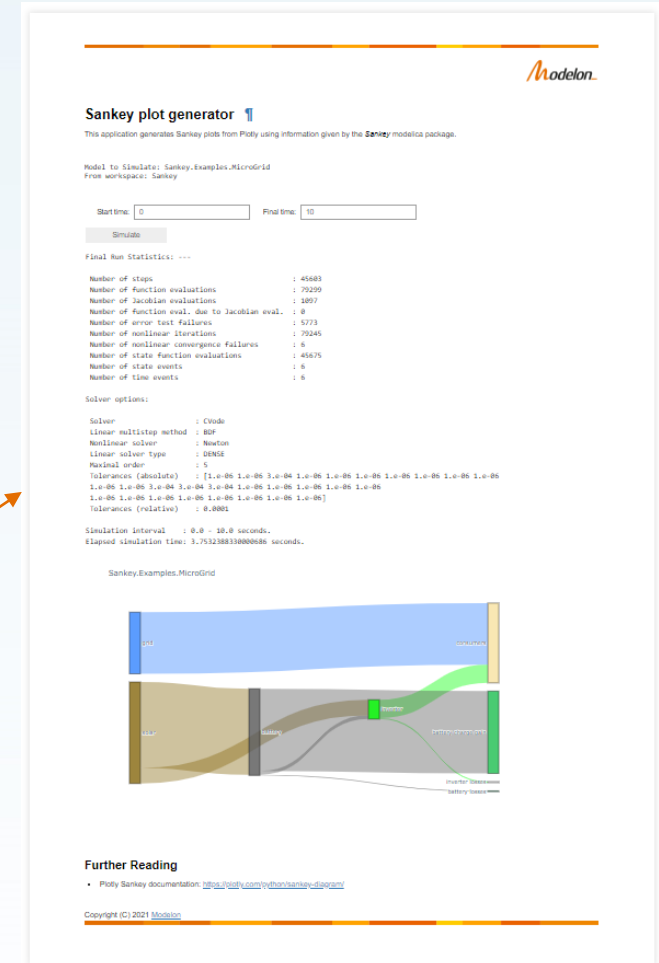
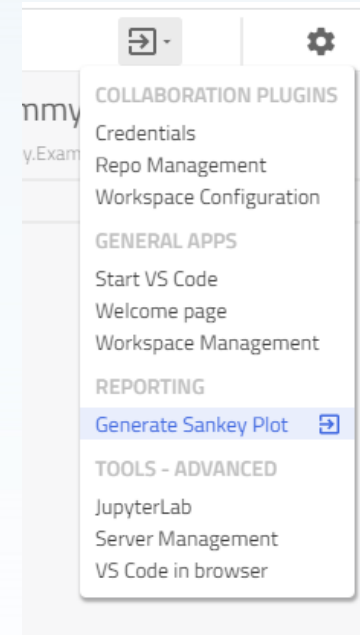
display(widgets.HBox([w_starttime, w_finaltime]))

w_simulate = widgets.Button(
    description="Simulate",
```

Example: Voila app

Deploy:

- Launch the app from the apps dropdown menu
- Model and Workspace information is carried over

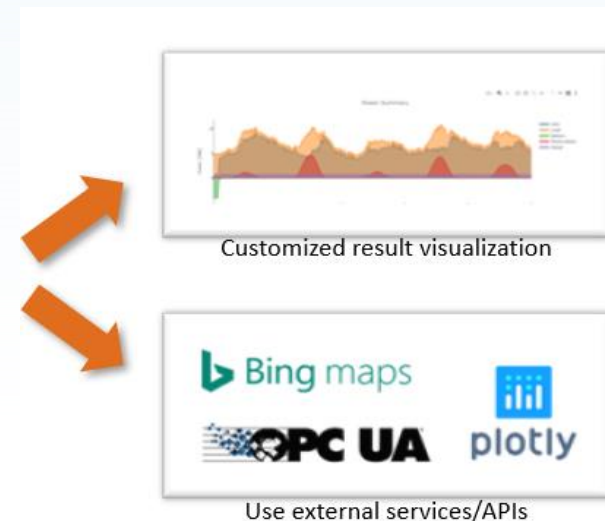


Demo – Python Dashboarding (Voila)

Traditional Web Applications (JS + HTML + CSS)

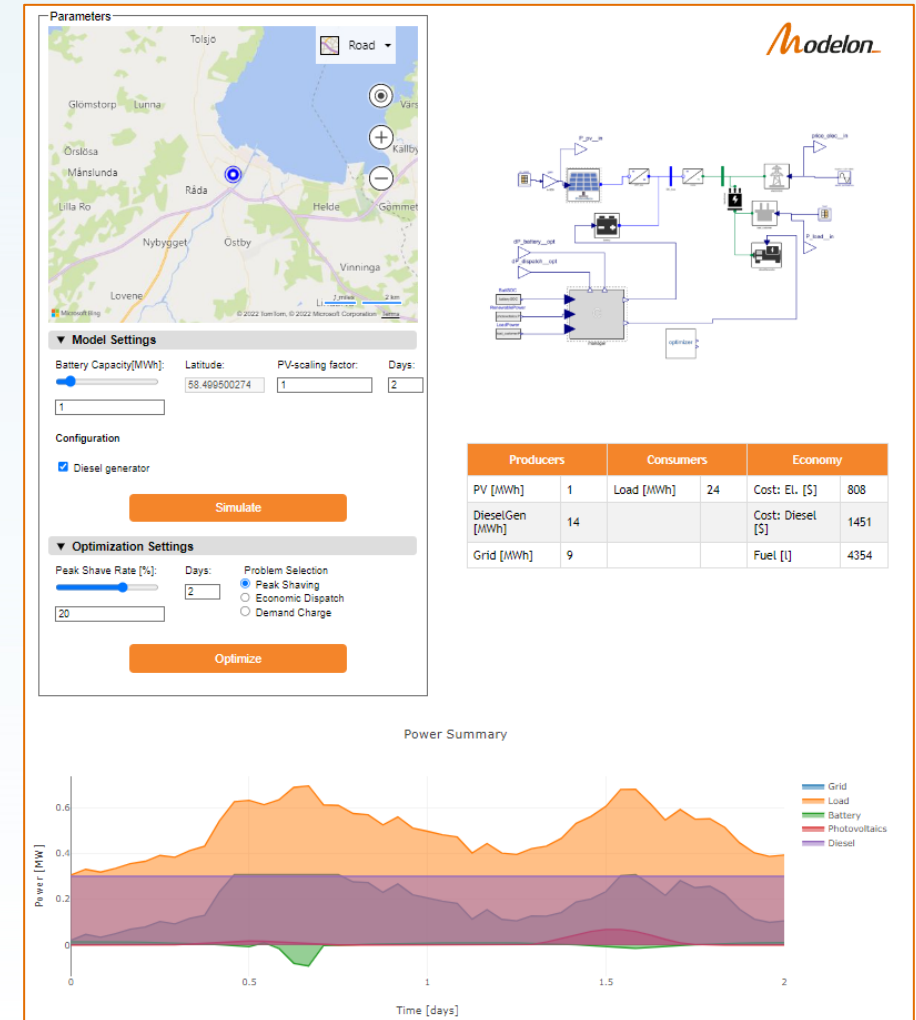
Full-blown web applications

- Technology allows for creation of full-blown web applications
- Provides full flexibility
- Requires more effort and specific competence
- Example use-cases:
 - Customized/Interactive result inspection
 - Integrate with external services APIs



Example: Microgrid Dashboard

- Map-Widget to fetch geographical data to the model
- User only presented with a relevant few parameters and settings
 - Problem mode
 - System configuration
 - Sizing information



Conclusion

- Custom interfaces have a wide range of use cases and benefits including:
 - Streamlining common workflows
 - Free up model developer resources
 - Removing the need for users to learn a new simulation tool
- Utilizing the Modelon Impact API + wrapper libraries we can create such interfaces
- Make it even simpler by utilizing App-Mode!

Modelon

Accurate Simulations. Better Decisions.