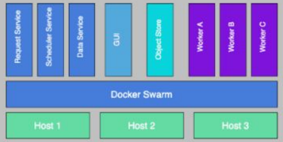
OWP | OFFICE OF WATER PREDICTION

Leveraging the Distributed Model on Demand Platform to Work With Community Models in the Next Generation Water Resources Modeling Framework

Robert Bartel, Austin Raney, Christopher Tubbs, Nels J. Frazier, Trey Flowers, Shengting Cui

NWS/Office of Water Prediction/National Water Center; Lynker Technologies, LLC

1 Intro - What is DMOD?



- Prototype platform for specialized, scalable compute environments (especially with ngen)
- **Key Feature: abstract and manage compute infrastructure and environments**

OPEN

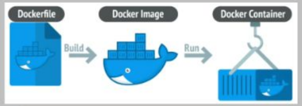
2 What's Needed for ngen

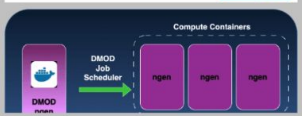
Language Support

Dependencies

OPEN

3 Using Docker Containers





OPEN

4 Missing BMI Modules

OPEN

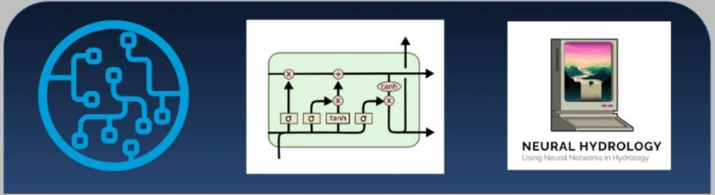
5 Guided Customization

Customization Directory

- Special Git-ignored customization

OPEN

6 An Example: A Forked LSTM BMI Module



OPEN

7 Summary and Future

What DMOD Does

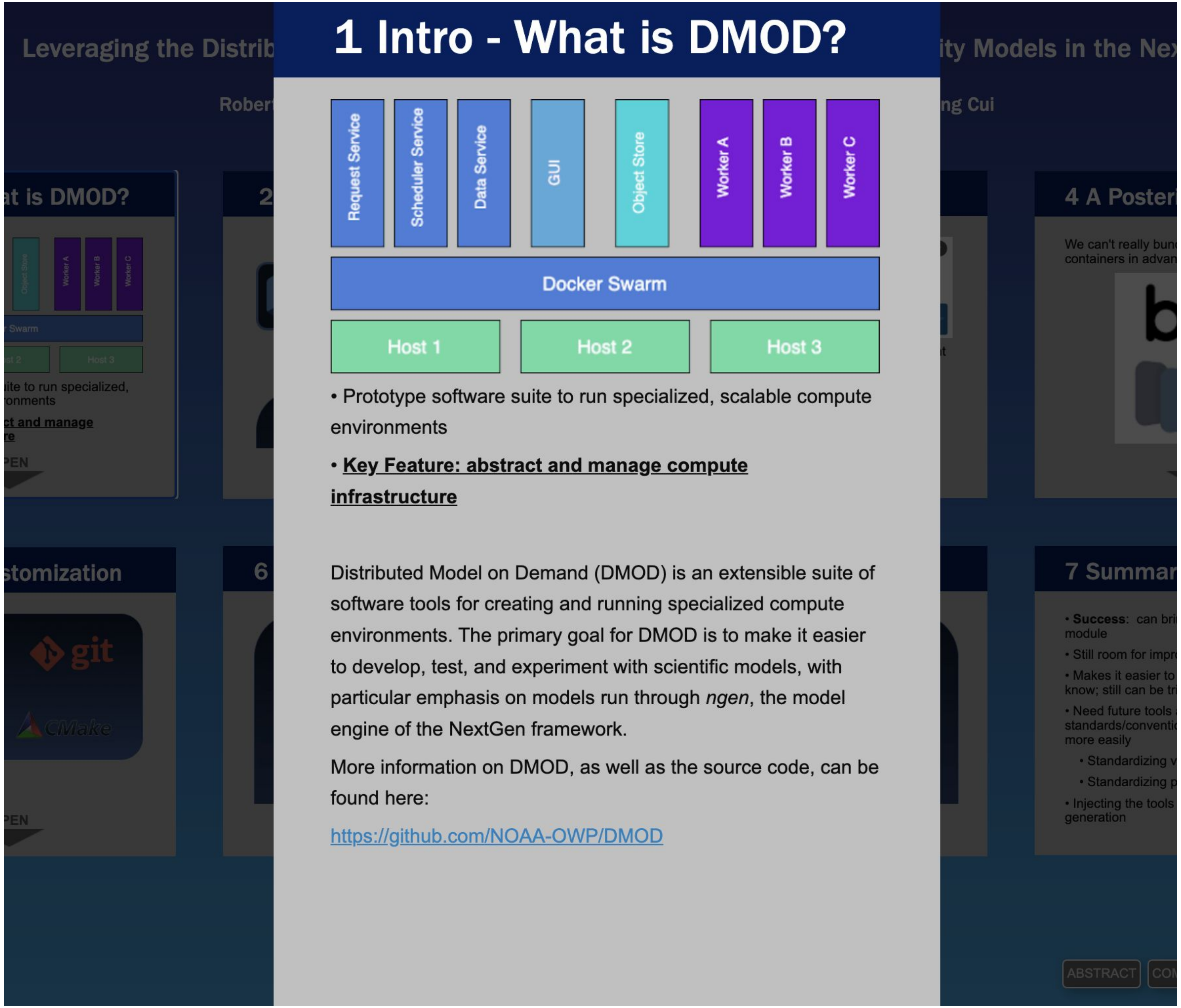
- Automates performance of typical steps for creating consistent compute environments for ngen via Docker images
- Enables local customization to incorporate external BMI modules into image builds without modifying distributed source code

Future Ideas

- Still room for improvements
  - Both in and beyond DMOD

OPEN

ABSTRACT | COMMENT | CONTACT | GET POSTER



1 Intro - What is DMOD?

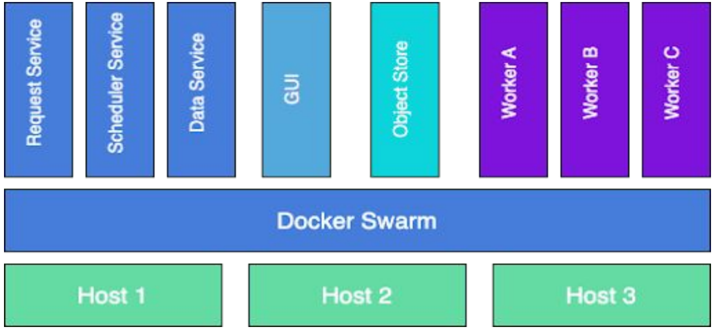
- Prototype software suite to run specialized, scalable compute environments
- **<u>Key Feature: abstract and manage compute infrastructure</u>**

Distributed Model on Demand (DMOD) is an extensible suite of software tools for creating and running specialized compute environments. The primary goal for DMOD is to make it easier to develop, test, and experiment with scientific models, with particular emphasis on models run through *ngen*, the model engine of the NextGen framework.

More information on DMOD, as well as the source code, can be found here:

https://github.com/NOAA-OWP/DMOD

1 Intro - What is DMOD?



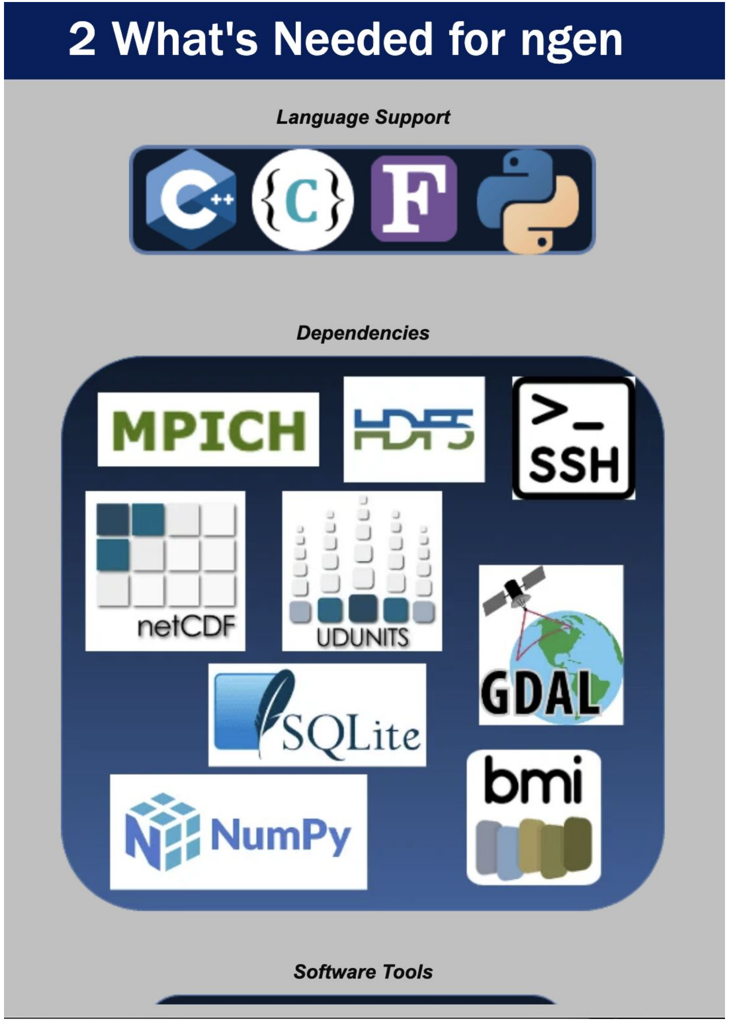
- Prototype software suite to run specialized, scalable compute environments

- **Key Feature: abstract and manage compute infrastructure**

Distributed Model on Demand (DMOD) is an extensible suite of software tools for creating and running specialized compute environments. The primary goal for DMOD is to make it easier to develop, test, and experiment with scientific models, with particular emphasis on models run through ngen, the model engine of the NextGen framework.

More information on DMOD, as well as the source code, can be found here:

https://github.com/NOAA-OWP/DMOD

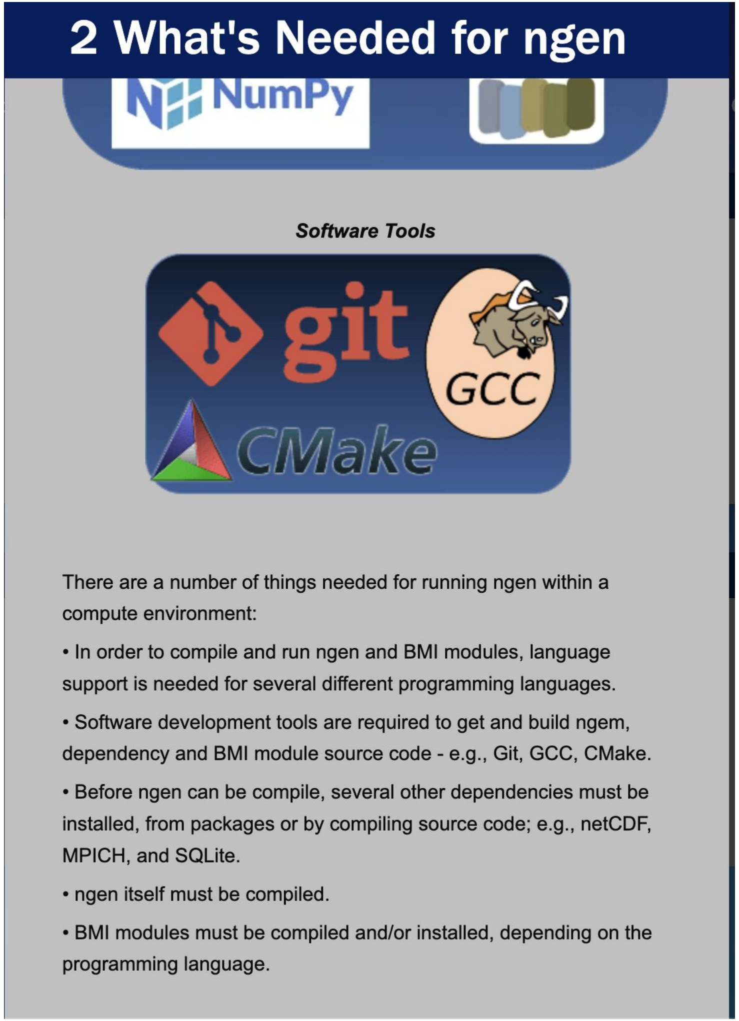


2 What's Needed for ngen

Language Support

Dependencies

Software Tools



2 What's Needed for ngen

Software Tools

There are a number of things needed for running ngen within a compute environment:

- In order to compile and run ngen and BMI modules, language support is needed for several different programming languages.
- Software development tools are required to get and build ngem, dependency and BMI module source code - e.g., Git, GCC, CMake.
- Before ngen can be compile, several other dependencies must be installed, from packages or by compiling source code; e.g., netCDF, MPICH, and SQLite.
- ngen itself must be compiled.
- BMI modules must be compiled and/or installed, depending on the programming language.

2 What’s Needed for ngen

There are a number of things needed for running ngen within a compute environment:

- In order to compile and run ngen and BMI modules, language support is needed for several different programming languages.

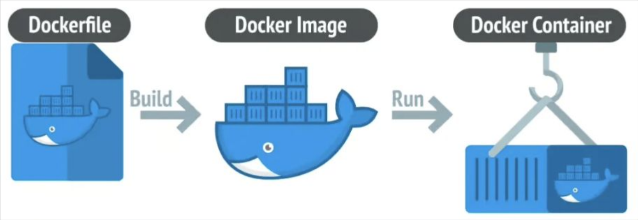
- Software development tools are required to get and build ngem, dependency and BMI module source code - e.g., Git, GCC, CMake.

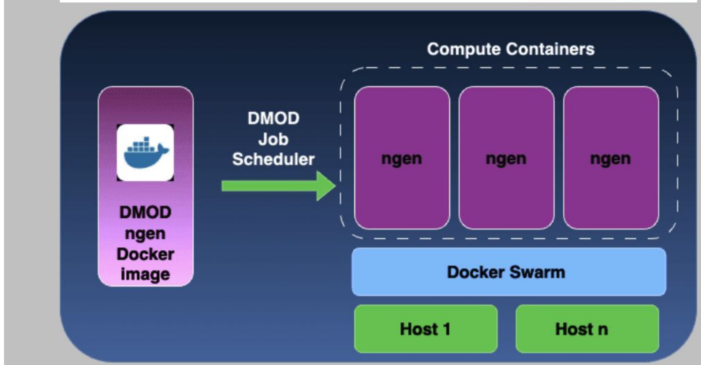
- Before ngen can be compile, several other dependencies must be installed, from packages or by compiling source code; e.g., netCDF, MPICH, and SQLite.

- ngen itself must be compiled.

- BMI modules must be compiled and/or installed, depending on the programming language.

3 Using Docker Containers

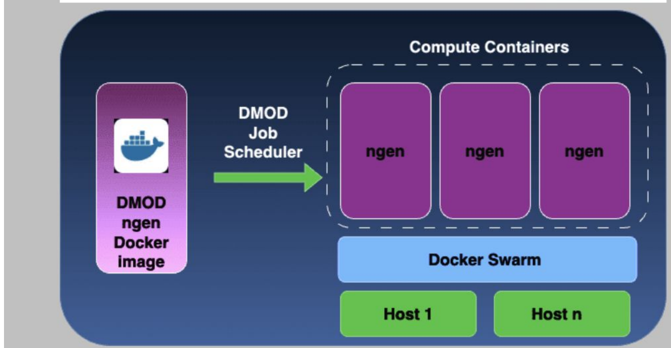




DMOD's Approach

- Maintains compute environment infrastructure as source code via *Dockerfiles*
- Includes custom tools to locally build Docker *images* from these Dockerfiles
- Has services that execute ngen by starting Docker *containers* from those images

3 Using Docker Containers



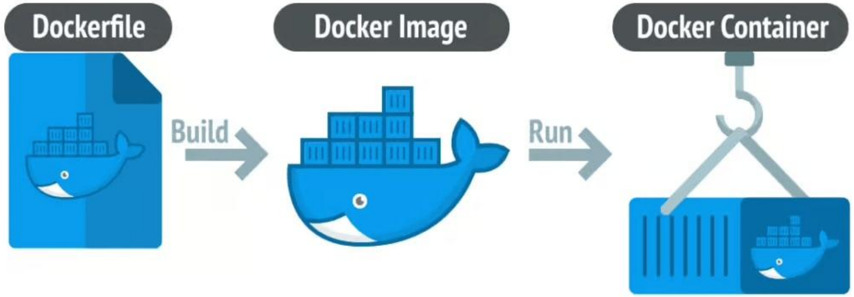
DMOD's Approach

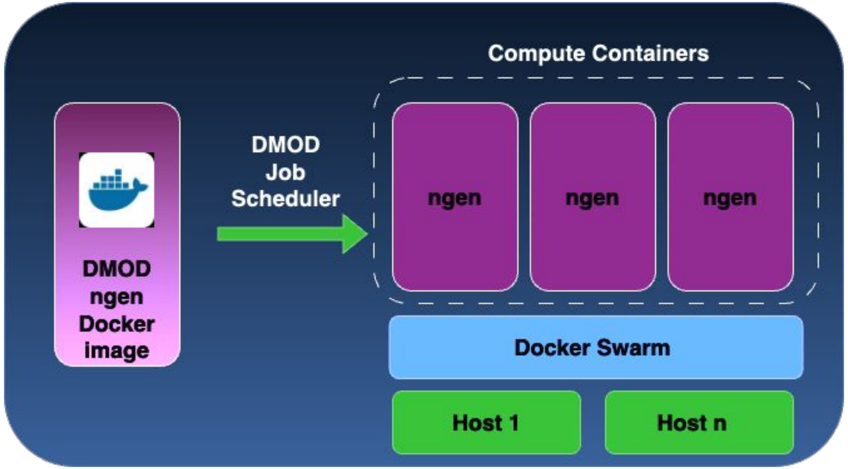
- Maintains compute environment infrastructure as source code via *Dockerfiles*
- Includes custom tools to locally build Docker *images* from these Dockerfiles
- Has services that execute ngen by starting Docker *containers* from those images

Key Benefits

- Users don't need to figure out how to compile ngen
- Users get consistent, re-creatable compute environments

3 Docker Containers



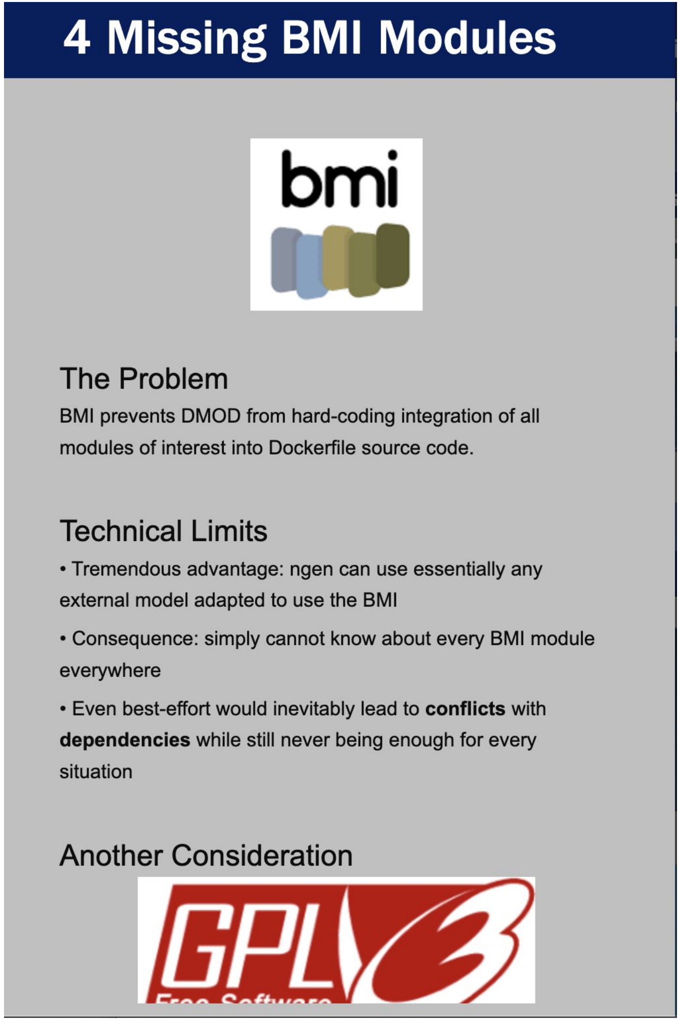


DMOD's Approach

- Maintains compute environment infrastructure as source code via Dockerfiles
- Includes custom tools to locally build Docker images from these Dockerfiles
- Has services that execute ngen by starting Docker containers from those images

Key Benefits

- Users don’t need to figure out how to compile ngen
- Users get consistent, re-creatable compute environments



4 Missing BMI Modules

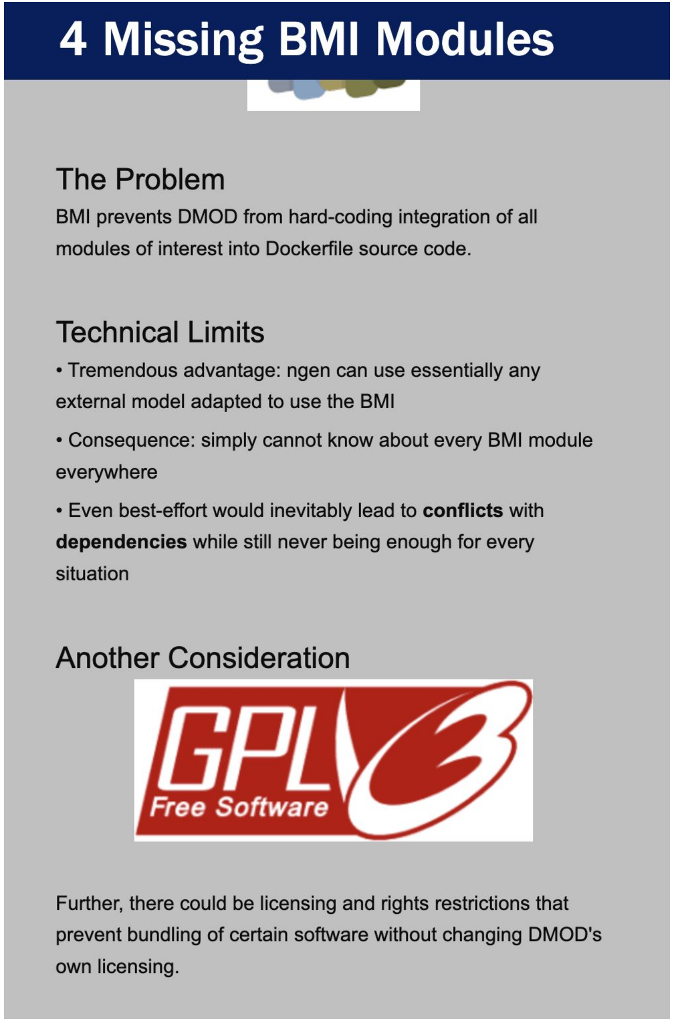
The Problem

BMI prevents DMOD from hard-coding integration of all modules of interest into Dockerfile source code.

Technical Limits

- Tremendous advantage: ngen can use essentially any external model adapted to use the BMI
- Consequence: simply cannot know about every BMI module everywhere
- Even best-effort would inevitably lead to **conflicts** with **dependencies** while still never being enough for every situation

Another Consideration



4 Missing BMI Modules

The Problem

BMI prevents DMOD from hard-coding integration of all modules of interest into Dockerfile source code.

Technical Limits

- Tremendous advantage: ngen can use essentially any external model adapted to use the BMI
- Consequence: simply cannot know about every BMI module everywhere
- Even best-effort would inevitably lead to **conflicts** with **dependencies** while still never being enough for every situation

Another Consideration

Further, there could be licensing and rights restrictions that prevent bundling of certain software without changing DMOD's own licensing.

4 Limits on Unlimited

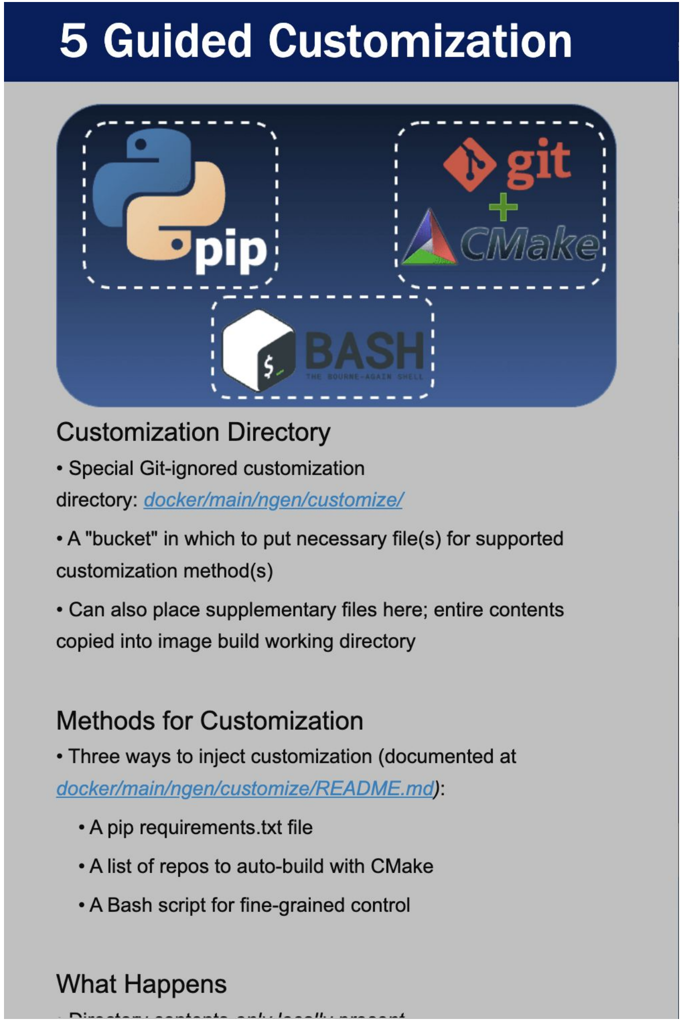
The Problem
BMI prevents DMOD from hard-coding integration of all modules of interest into Dockerfile source code.

Technical Limits

- Tremendous advantage: ngen can use essentially any external model adapted to use the BMI
- Consequence: simply cannot know about every BMI module everywhere
- Even best-effort would inevitably lead to conflicts with dependencies while still never being enough for every situation

Another Consideration

Further, there could be licensing and rights restrictions that prevent bundling of certain software without changing DMOD's own licensing.



5 Guided Customization

Customization Directory

- Special Git-ignored customization directory: docker/main/ngen/customize/
- A "bucket" in which to put necessary file(s) for supported customization method(s)
- Can also place supplementary files here; entire contents copied into image build working directory

Methods for Customization

- Three ways to inject customization (documented at *docker/main/ngen/customize/README.md*):
  - A pip requirements.txt file
  - A list of repos to auto-build with CMake
  - A Bash script for fine-grained control

What Happens

- Directory contents *only locally present*

5 Guided Customization

BASH
THE BOURNE-AGAIN SHELL

Customization Directory

- Special Git-ignored customization directory: docker/main/ngen/customize/
- A "bucket" in which to put necessary file(s) for supported customization method(s)
- Can also place supplementary files here; entire contents copied into image build working directory

Methods for Customization

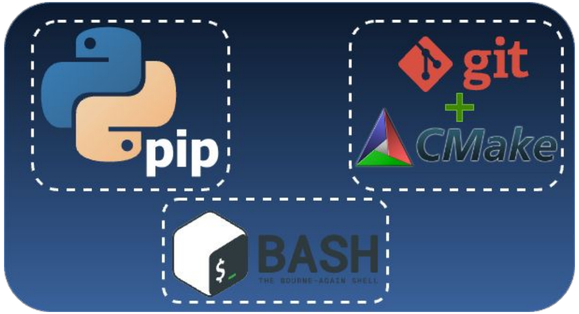
- Three ways to inject customization (documented at *docker/main/ngen/customize/README.md*):
  - A pip requirements.txt file
  - A list of repos to auto-build with CMake
  - A Bash script for fine-grained control

What Happens

- Directory contents *only locally present*

  - *Not committed to the repository*
- When present, these get used automatically by DMOD image build tools
- They trigger execution of the customization steps at a specific stage of the ngen Docker image build

5 Enable Customization



Customization Directory
- Special Git-ignored customization directory: docker/main/ngen/customize/

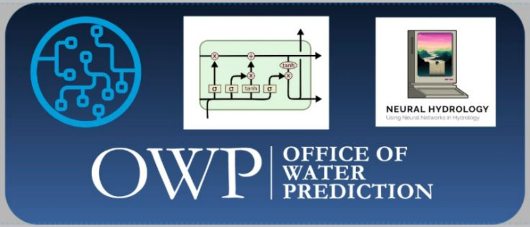
- A "bucket" in which to put necessary file(s) for supported customization method(s)

- Can also place supplementary files here; entire contents copied into image build working directory

Methods for Customization
- Three ways to inject customization (documented at docker/main/ngen/customize/README.md):
    - A pip requirements.txt file
    - A list of repos to auto-build with CMake
    - A Bash script for fine-grained control

What Happens
- Directory contents only locally present
    - Not committed to the repository
- When present, these get used automatically by DMOD image build tools
- They trigger execution of the customization steps at a specific stage of the ngen Docker image build

6 An Example: A Forked LSTM BMI Module


NEURAL HYDROLOGY
Using Neural Networks in Hydrology
OWP
OFFICE OF WATER PREDICTION

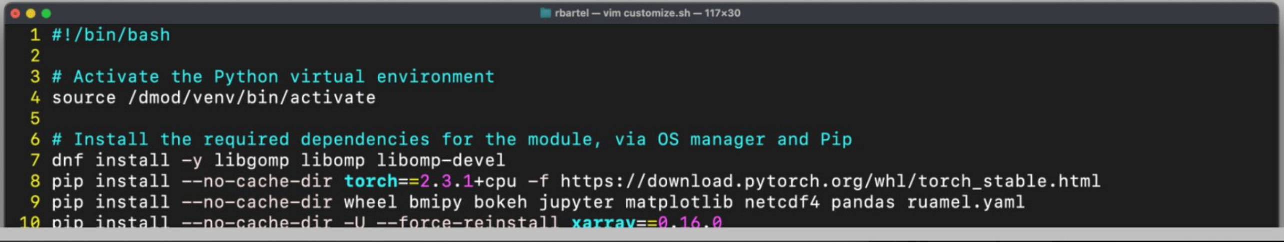
To illustrate, consider an LSTM BMI module.

The source code can be found at https://github.com/robertbartel/owp_lstm

Important: not pre-integrated into Docker image, not (directly) from OWP's public repo.

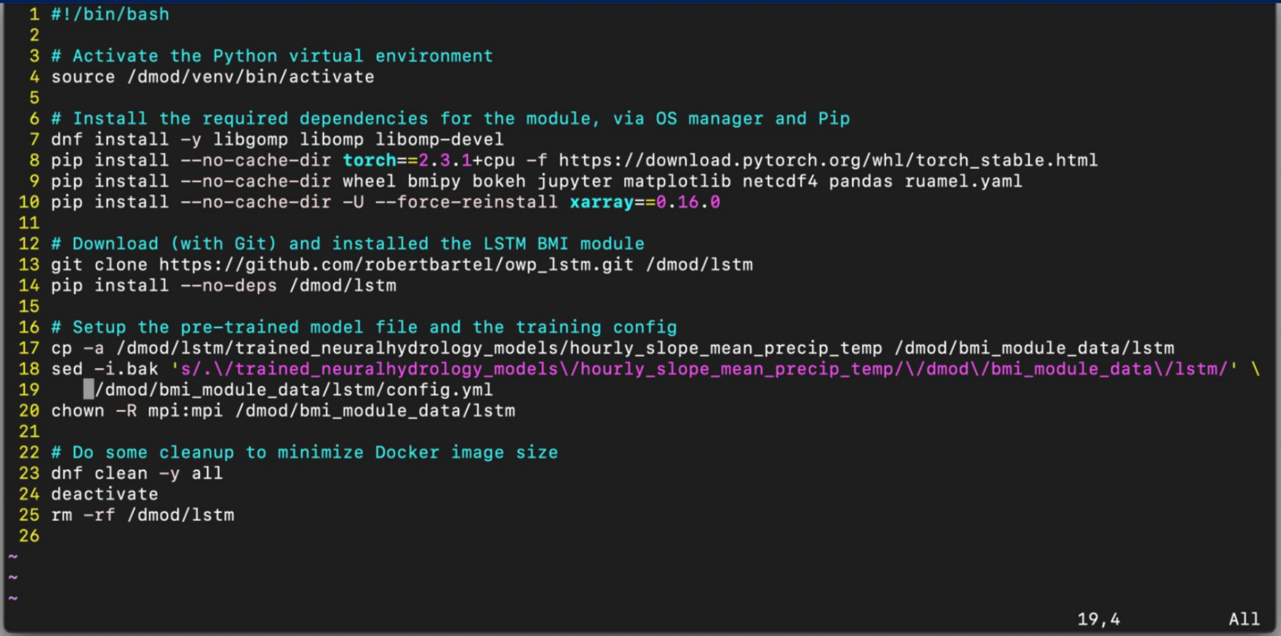
The Steps

Create a local *docker/main/ngen/customize/customize.sh* file:



```
#!/bin/bash

# Activate the Python virtual environment
source /dmod/venv/bin/activate

# Install the required dependencies for the module, via OS manager and Pip
dnf install -y libgomp libomp libomp-devel
pip install --no-cache-dir torch==2.3.1+cpu -f https://download.pytorch.org/whl/torch_stable.html
pip install --no-cache-dir wheel bmipy bokeh jupyter matplotlib netcdf4 pandas ruamel.yaml
pip install --no-cache-dir -U --force-reinstall xarray==0.16.0
```

6 An Example: A Forked LSTM BMI Module



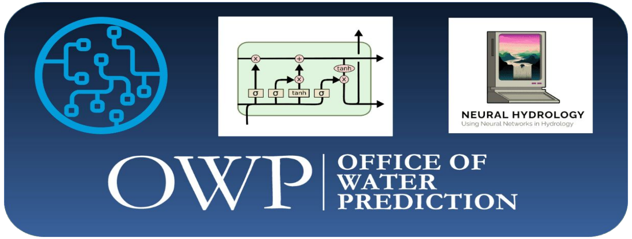
```
#!/bin/bash

# Activate the Python virtual environment
source /dmod/venv/bin/activate

# Install the required dependencies for the module, via OS manager and Pip
dnf install -y libgomp libomp libomp-devel
pip install --no-cache-dir torch==2.3.1+cpu -f https://download.pytorch.org/whl/torch_stable.html
pip install --no-cache-dir wheel bmipy bokeh jupyter matplotlib netcdf4 pandas ruamel.yaml
pip install --no-cache-dir -U --force-reinstall xarray==0.16.0

# Download (with Git) and installed the LSTM BMI module
git clone https://github.com/robertbartel/owp_lstm.git /dmod/lstm
pip install --no-deps /dmod/lstm

# Setup the pre-trained model file and the training config
cp -a /dmod/lstm/trained_neuralhydrology_models/hourly_slope_mean_precip_temp /dmod/bmi_module_data/lstm
sed -i.bak 's/.\/trained_neuralhydrology_models\/hourly_slope_mean_precip_temp/\/dmod\/bmi_module_data\/lstm/' \
    /dmod/bmi_module_data/lstm/config.yml
chown -R mpi:mpi /dmod/bmi_module_data/lstm

# Do some cleanup to minimize Docker image size
dnf clean -y all
deactivate
rm -rf /dmod/lstm

```

Then rebuild the Docker image using standard DMOD helper script:

```
./scripts/control_stack.sh --build-args "ngen" main build push
```

With this, we can run LSTM prediction for 1 month on about 18,000 catchments (VPU01) in about 16 minutes. This was on a PC with 13th Gen Intel i7 processor, using 16 of its cores.

6 An Example: A Forked LSTM BMI Module



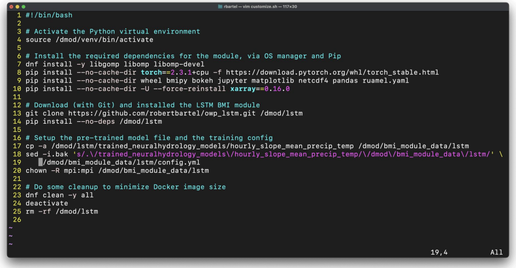
To illustrate, consider an LSTM BMI module.

The source code can be found at https://github.com/robertbartel/owp_lstm

Important: not pre-integrated into Docker image, not (directly) from OWP's public repo.

The Steps

Create a local docker/main/ngen/customize/customize.sh file:



```
#!/bin/bash

# Activate the Python virtual environment
source /dmod/venv/bin/activate

# Install the required dependencies for the module, via OS manager and Pip
dnf install -y libgomp libomp libomp-devel
pip install --no-cache-dir torch==2.3.1+cpu -f https://download.pytorch.org/whl/torch_stable.html
pip install --no-cache-dir wheel bmipy bokeh jupyter matplotlib netcdf4 pandas ruamel.yaml
pip install --no-cache-dir -U --force-reinstall xarray==0.16.0

# Download (with Git) and installed the LSTM BMI module
git clone https://github.com/robertbartel/owp_lstm.git /dmod/lstm
pip install --no-deps /dmod/lstm

# Setup the pre-trained model file and the training config
cp -a /dmod/lstm/trained_neuralhydrology_models/hourly_slope_mean_precip_temp /dmod/bmi_module_data/lstm
sed -i.bak 's/.\/trained_neuralhydrology_models\/hourly_slope_mean_precip_temp/\/dmod\/bmi_module_data\/lstm/' \
   /dmod/bmi_module_data/lstm/config.yml
chown -R mpi:mpi /dmod/bmi_module_data/lstm

# Do some cleanup to minimize Docker image size
dnf clean -y all
deactivate
rm -rf /dmod/lstm

~
~
~
                                                                                          19,4          All
```

Then rebuild the Docker image using standard DMOD helper script:

```
   ./scripts/control_stack.sh
--build-args "ngen" main build push
```

With this, we can run LSTM prediction for 1 month on about 18,000 catchments (VPU01) in about 16 minutes. This was on a PC with 13th Gen Intel i7 processor, using 16 of its cores.

7 Summary and Future

What DMOD Does

- Automates performance of typical steps for creating consistent compute environments for ngen via Docker images
- Enables local customization to incorporate external BMI modules into image builds without modifying distributed source code

Future Ideas

- Still room for improvements
  - Both in and beyond DMOD
  - More interoperable ngen+BMI ecosystem
- This makes it easier for a model *you* know
  - Still can be tricky for one “off the shelf”
  * Tried to incorporate the SUMMA module
  - Could not in time constraints
    - Troubles independent of DMOD
- Integratration with the tools for BMI init config generation would be beneficial
- Need more tools and standards/conventions to “package” models more effectively and consistently
  - Standardizing versions and dependencies
  - Standardizing builds and installations
  - Standardizing paths
- Consider separating development of ngen Docker images into repo focused on them

7 Summary and Future

What DMOD Does

• Automates performance of typical steps for creating consistent compute environments for ngen via Docker images

• Enables local customization to incorporate external BMI modules into image builds without modifying distributed source code

Future Ideas

- Still room for improvements
  - Both in and beyond DMOD
  - More interoperable ngen+BMI ecosystem
- This makes it easier for a model you know
  - Still can be tricky for one “off the shelf”
  - Tried to incorporate the SUMMA module
  - Could not in time constraints
    - Troubles independent of DMOD
- Integration with the tools for BMI init config generation would be beneficial
- Need more tools and standards/conventions to “package” models more effectively and consistently
  - Standardizing versions and dependencies
  - Standardizing builds and installations
  - Standardizing paths
- Consider separating development of ngen Docker images into repo focused on them