Implementation of WRF as a Service in the AWS Cloud

Don Morton
Boreal Scientific Computing, Fairbanks, Alaska, USA

Jamie Wolff, Kate Fossell, John Halley Gotway, Michael Kavulich, Jr, Michelle Harrold
Developmental Testbed Center, NCAR, Boulder, Colorado, USA

Acknowledgements
DTC Visitor Program
NOAA CSTAR NA16NWS4680006

Boreal Scientific Computing LLC

DTC Developmental Testbed Center

Abstract / Outline

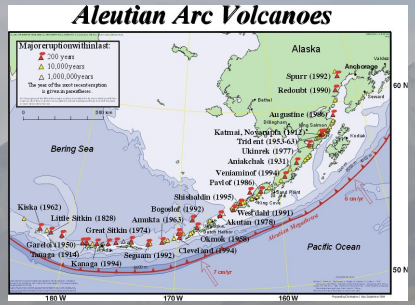
- Goal - support easy and flexible access to numerical weather prediction environments
- Construction of underlying “plumbing” and overlying abstractions for execution of WRF components in the AWS cloud
- Evolving implementation is based on deployment of DTC’s NWP Docker containers in the AWS Elastic Container Services (ECS)
- Presentation outlines motivations, vision, methods and current status

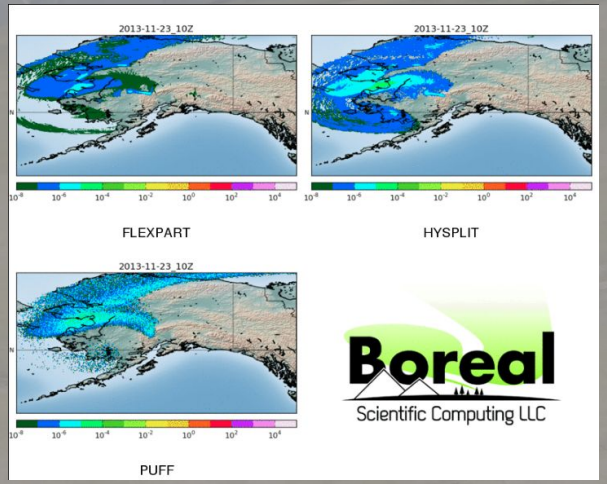
Boreal
Scientific Computing LLC

Scenario - volcanic ash transport

- Volcano eruptions tend to be one-time, unscheduled events
- Figuring out where the ash will go is a time-critical event, with financial impacts in hundreds of billions USD in air transport
- The dream is to launch an ensemble of regional NWP forecasts when eruption is imminent, and use the met data to drive plume forecasts
- Need to generate custom domain and parameter configurations quickly
- Need to launch simulations in a “burst” on dedicated computational resources
- Need for automation - the “pajama factor” - an alert at 2am triggers these simulations so that by the time we’re out of our jammies and at work at 5am, we already have preliminary forecasts

Woodsen Saunders, via alaskapublic.org



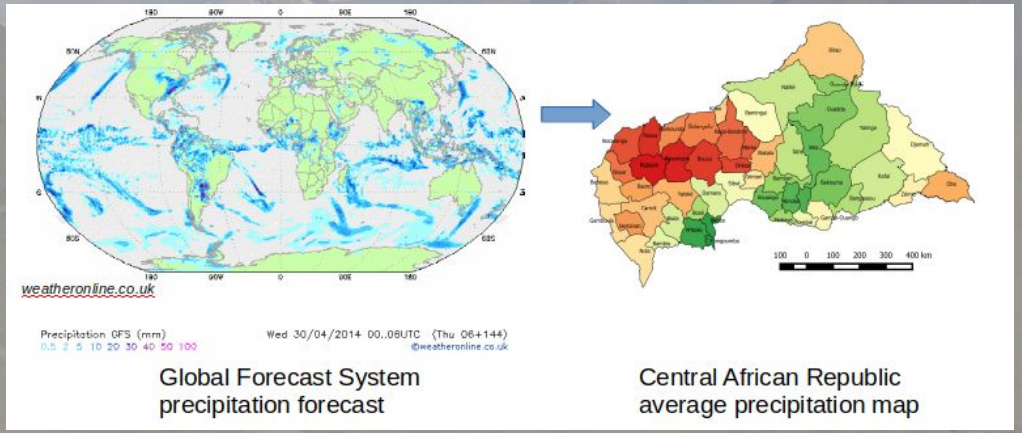


Boreal
Scientific Computing LLC

Inspiration: Peter Webley, University of Alaska Fairbanks, Alaska Volcano Observatory (in the past)

Scenario - staging relief resources prior to severe weather

- There was an interest in monitoring large scale synoptics over central Africa and, in times of potential trouble to launch a regional, customised set of NWP forecasts for more targeted assessment
- Again, the need was sporadic, but immediate, and model setup would be unknown until the last minute
- The initial thought was to set up an already busy supercomputer to be “preempted” in times of need, but this was not seen as reliable



Global Forecast System precipitation forecast

Central African Republic average precipitation map

Inspiration: Emily Niebuhr, Alaska Region NWS, United Nations World Food Programme, Rome

Boreal Scientific Computing LLC

The Holy Grail - long-term motivations

We want to be able to launch numerical weather simulations so that

- the tedious details are hidden
- they can be based on custom domain, parameter and workflow configurations
- they can start running immediately, and on short-notice (unscheduled)

Some target scenarios include

- emergency response to track ash after volcanic eruption
- proactive staging of relief resources for severe weather events
- experimental variations of existing operational forecasts
- “back of the envelope” research experiments
- last-minute simulations before a conference presentation!

Boreal
Scientific Computing LLC

The Holy Grail - long-term motivations

We want to be able to launch numerical weather simulations so that

- the tedious details a
- they can be based
- they can start running immediately, and on short-notice (unscheduled)

ations

The target audience - atmospheric scientists lacking the skills and/or patience to deal with the many geek aspects of NWP

Some target scenarios include

- emergency response to track ash after volcanic eruption
- proactive staging of relief resources for severe weather events
- experimental variations of existing operational forecasts
- “back of the envelope” research experiments
- last-minute simulations before a conference presentation!

Boreal
Scientific Computing LLC

Some requirements for achieving the vision

- Immediate access to sufficient computational resources, perhaps on a very infrequent basis
- Fine-grained control of NWP processes that can be arranged in novel and independent workflows
- Well-defined, robust APIs to facilitate coordination of the fine-grained NWP processes
- Stable environment - ideally each simulation gets its own dedicated resources, already fully tested in a non-changing operating environment
- ROI relative to existing solutions must be attractive

Boreal
Scientific Computing LLC

Satisfying the requirements

- Cloud-bursting paradigm - for most of the time resources may not be required, but when we do need them, no dilly dallying!
- Loosely-coupled, distributed services for creation of flexible, custom workflows
- Abstracting away the horrible, tedious details

Boreal
Scientific Computing LLC

Abstracting away the details

- Setting up and running WRF is hard - time-consuming and error-prone
  - Download, compile, maintain environment variables
  - Set up configuration files and input data in a strict environment
  - Additional complexities come into play when running on multiple processors on a shared system
- Many talented atmospheric scientists resist the use of NWP models for their work simply because of the time and effort required

Boreal
Scientific Computing LLC

Abstracting away the details

Excerpts from my notes when trying to set up WRF for a United Nations research group in Vienna in 2011

```
./configure

  25.  x86_64 Linux, gfortran compiler with gcc   (dmpar)

Basic nesting (option 1)

In configure.wrf

Removed the -ftree-loop-linear option from FCOPTIM  (Not sure I really needed to do this, but I did
need to at home, with a newer version of gfortran)


DM_FC   =   /dvlscratch/atm_new/MORTON_WRF/SupportingLibs/mpich2-1.5/bin/mpif90
DM_CC   =   /dvlscratch/atm_new/MORTON_WRF/SupportingLibs/mpich2-1.5/bin/mpicc -DMPI2_SUPPORT

Issues:

./compile em_real 2>&1 | tee compile.out

        ranlib
/dvlscratch/atm_new/MORTON_WRF/WRFV3.4-2012-11-30/WRFV3/external/RSL_LITE/librsl_lite.a )

make[3]: Entering directory
`/dvlscratch/atm_new/MORTON_WRF/WRFV3.4-2012-11-30/WRFV3/external/RSL_LITE'

/dvlscratch/atm_new/MORTON_WRF/SupportingLibs/mpich2-1.5/bin/mpicc -DFSEEKO64_OK  -w -O3 -c
-DLANDREAD_STUB -DDM_PARALLEL -DMAX_HISTORY=25  -c c_code.c

In file included from c_code.c:23:

rsl_lite.h:152: error: redefinition of typedef ‘MPI_Fint’

/dvlscratch/atm_new/MORTON_WRF/SupportingLibs/mpich2-1.5/include/mpi.h:508: note: previous
declaration of ‘MPI_Fint’ was here

make[3]: [c_code.o] Error 1 (ignored)
```

Boreal Scientific Computing LLC

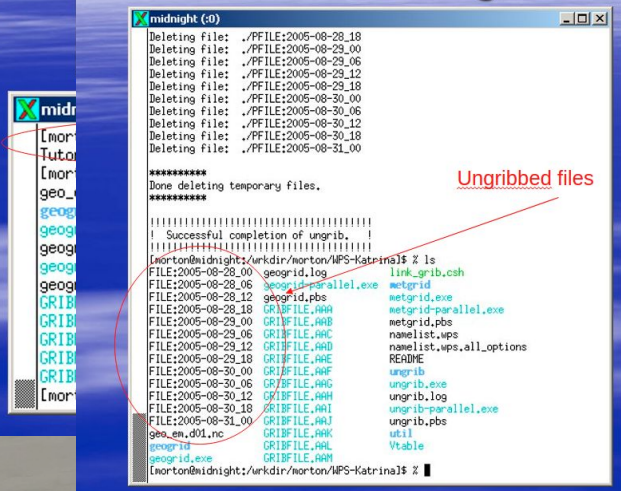
Abstracting away the details

Modify *namelist.wps*

Make Link to Appropriate Vtable

Make links to the input files

Run *ungrib.exe*



Boreal Scientific Computing LLC

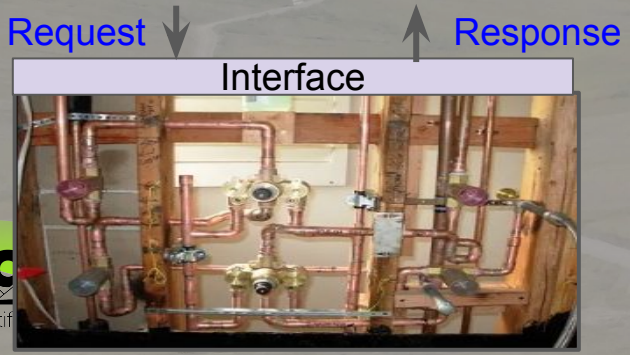
Some driving concepts

- The Grid
- Virtualization / containerization
- The ideal interface
- The “it’s gonna have to do for now” interface
- Containerization of NWP processes

Boreal
Scientific Computing LLC

Some driving concepts

- The Grid
- Virtualization / containerization
- The ideal interface
- The “it’s gonna have to do for r
- Containerization of NWP proce



At the turn of the century, with ubiquitous access to networked computing resources, the vision of a computational grid was promoted

- Like an electrical grid, we would “plug in” and get remote resources, not knowing or caring where they were actually coming from
- People would be able to run complex simulations in this grid via very simple interfaces, not worrying about the underlying complexities
- In a modern realization of this vision, imagine asking Alexa to run an NWP simulation based on a few custom specifications
- I suggest that all of the pieces for doing this are readily available and the challenge is in building the “plumbing” to put them all together, all while hiding the complexity from the user

Some driving concepts

- The Grid
- Virtualization / containerization
- The ideal interface
- The “it’s gonna have to do for now” interface
- Containerization of NWP processes

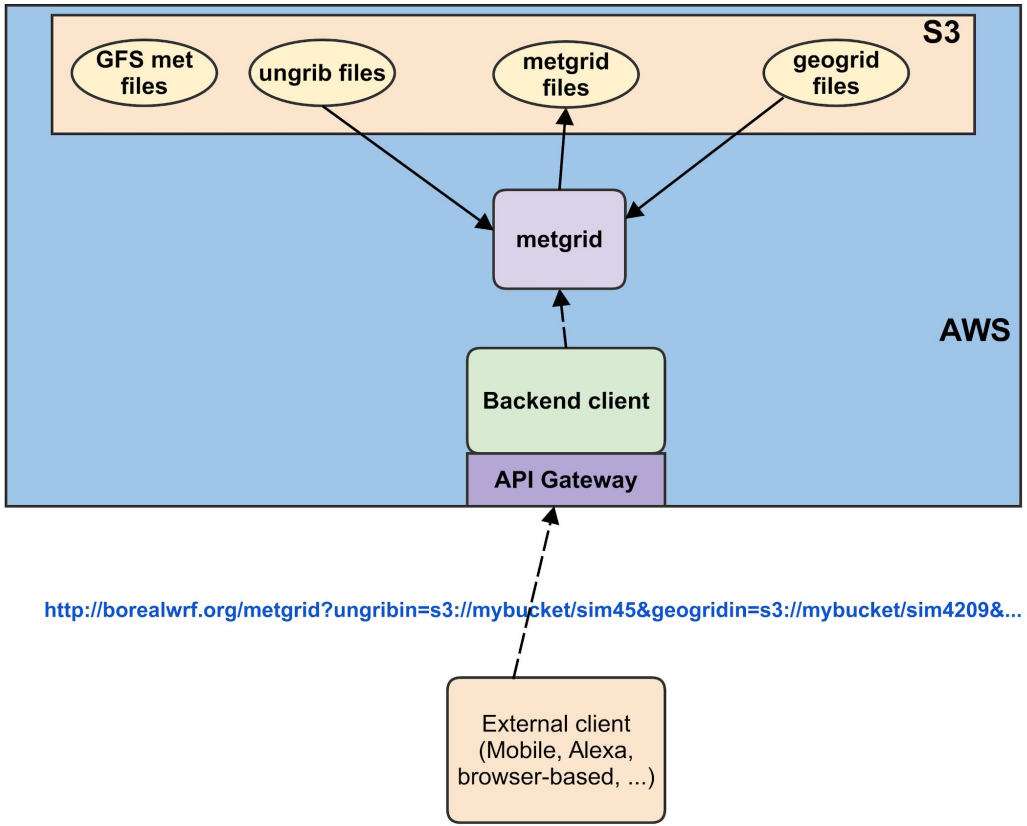
- For decades, virtual machines have allowed us to run one operating system on top of another, for many different reasons
- More recently, VMs and containers can be scripted and shared, providing all users with a standard **non-changing** computing platform

Boreal Scientific Computing LLC

Some driving conce

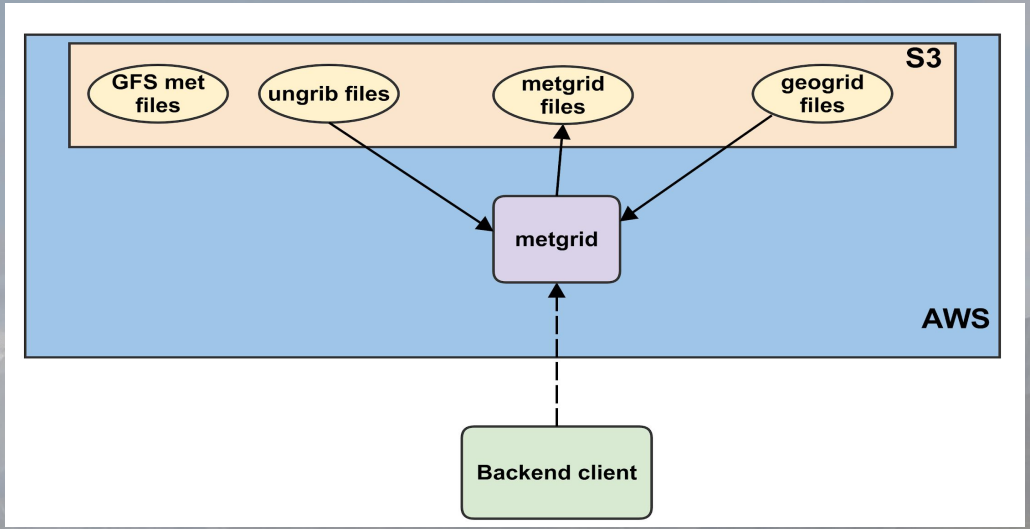
- The Grid
- Virtualization / contair
- The ideal interface
- The “it’s gonna have t
- Containerization of N

In this idealized scenario, launching a custom WRF simulation boils down to construction of appropriate URIs



Scientific Computing LLC

Some driving concepts



- The Grid
- Virtualization / containerization
- The ideal interface
- The “it’s gonna have to do for now” interface
- Containerization of NWP processes

The near-term goal is to understand and prototype a client-directed launching of a WRF service

Boreal Scientific Computing LLC

Some driving concepts

- The Grid
- Virtualization / containerization
- The ideal interface
- The “it’s gonna have to do for now” inte
- Containerization of NWP processes

- DTC has produced a set of Docker containers for NWP components
- Each is a self-contained system that includes everything necessary to run without requiring up-front setup
- Anybody can download and use these to run a rigorously-tested stable implementation of NWP tools, on any platform supporting Docker
- Availability of these containers means that the primary NWP service components are already installed and ready to use. All that’s needed is “plumbing” and directions for execution

Boreal
Scientific Computing LLC

The micro(haha) services paradigm

- AWS has great support for microservices (Lambda, ECS, Fargate) and it's a very common theme in use of cloud resources
  - Client makes a request, a service in the cloud quickly starts up and processes it, response is sent back to client.
- The problem is that even the simplest NWP processes are big and long-lasting, and can't be handled like typical microservices. Additionally, the responses are frequently huge files
- Ultimately, we have asynchronous processes (sometimes running for hours) that need to be handled through the use of synchronous processes

Boreal
Scientific Computing LLC

A micro(haha) service implementation

Client launches a service into AWS, keeps track of it for its duration

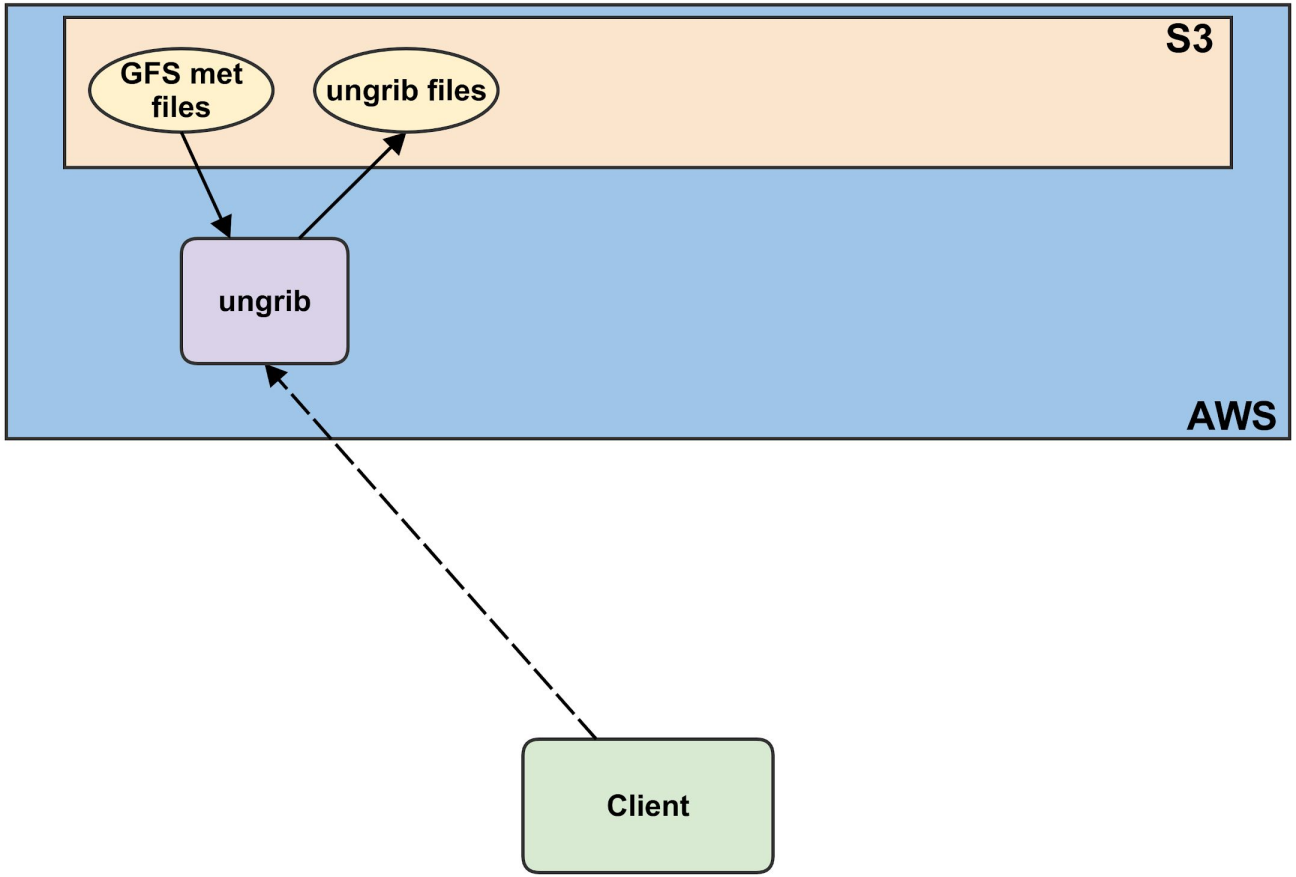
- Provide arguments to the service, including where (typically S3 objects) it should look for any input files
- Launch the service, and know how to query for ongoing status
- Periodically query the service for status
- Upon successful completion, know where (typically in a specified S3 location) to find all output
- Shut down the service

Boreal
Scientific Computing LLC

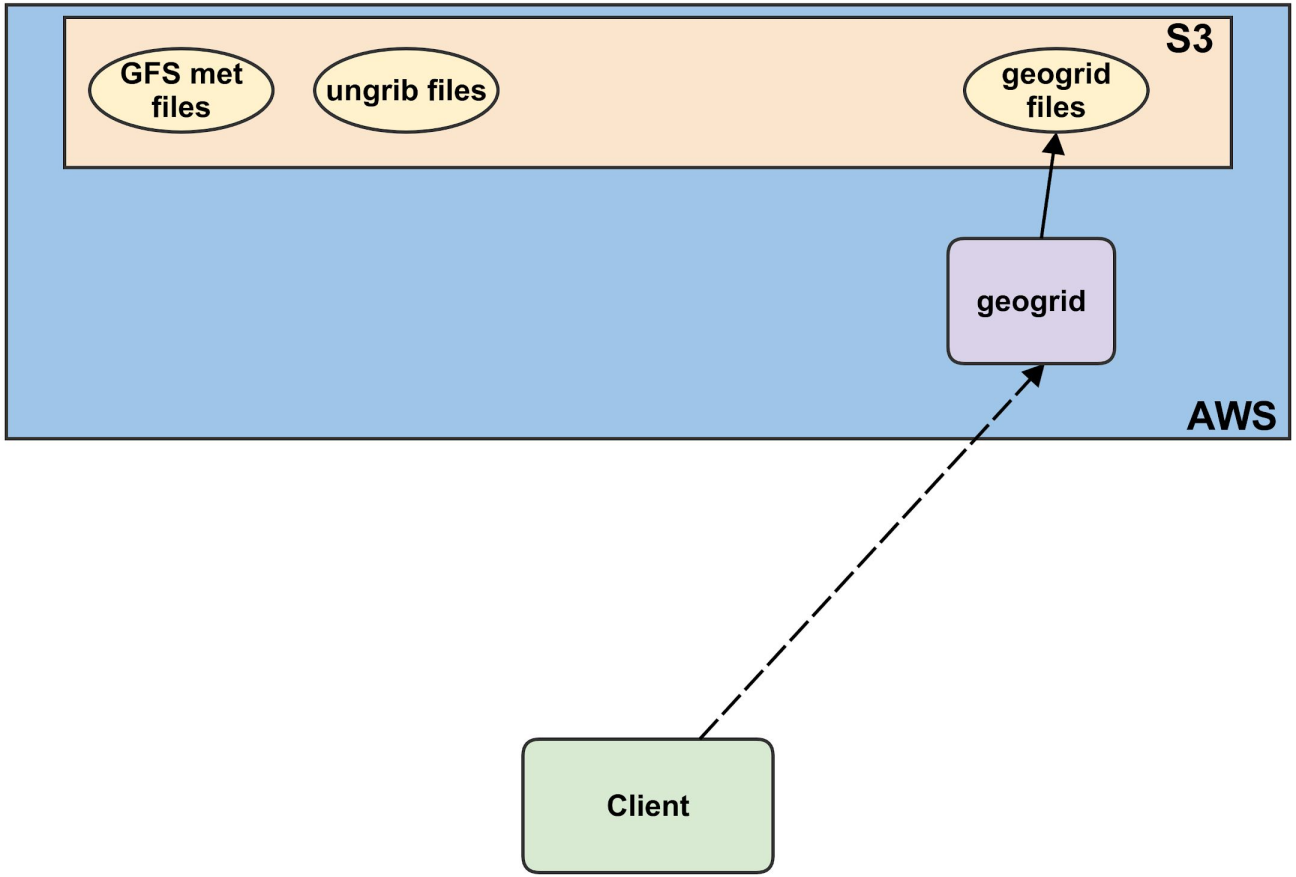
WPS services workflow

Client

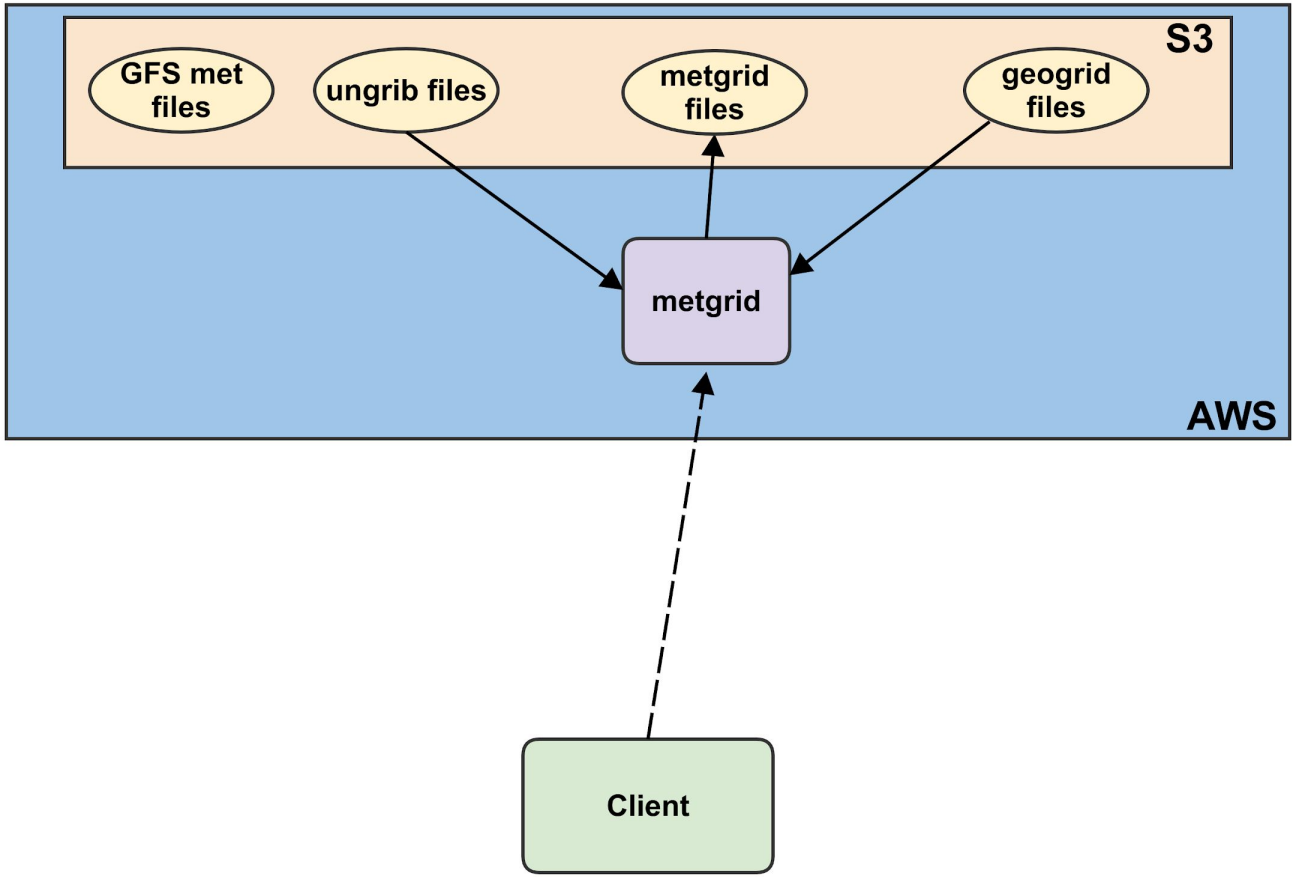
WPS services workflow



WPS services workflow



WPS services workflow

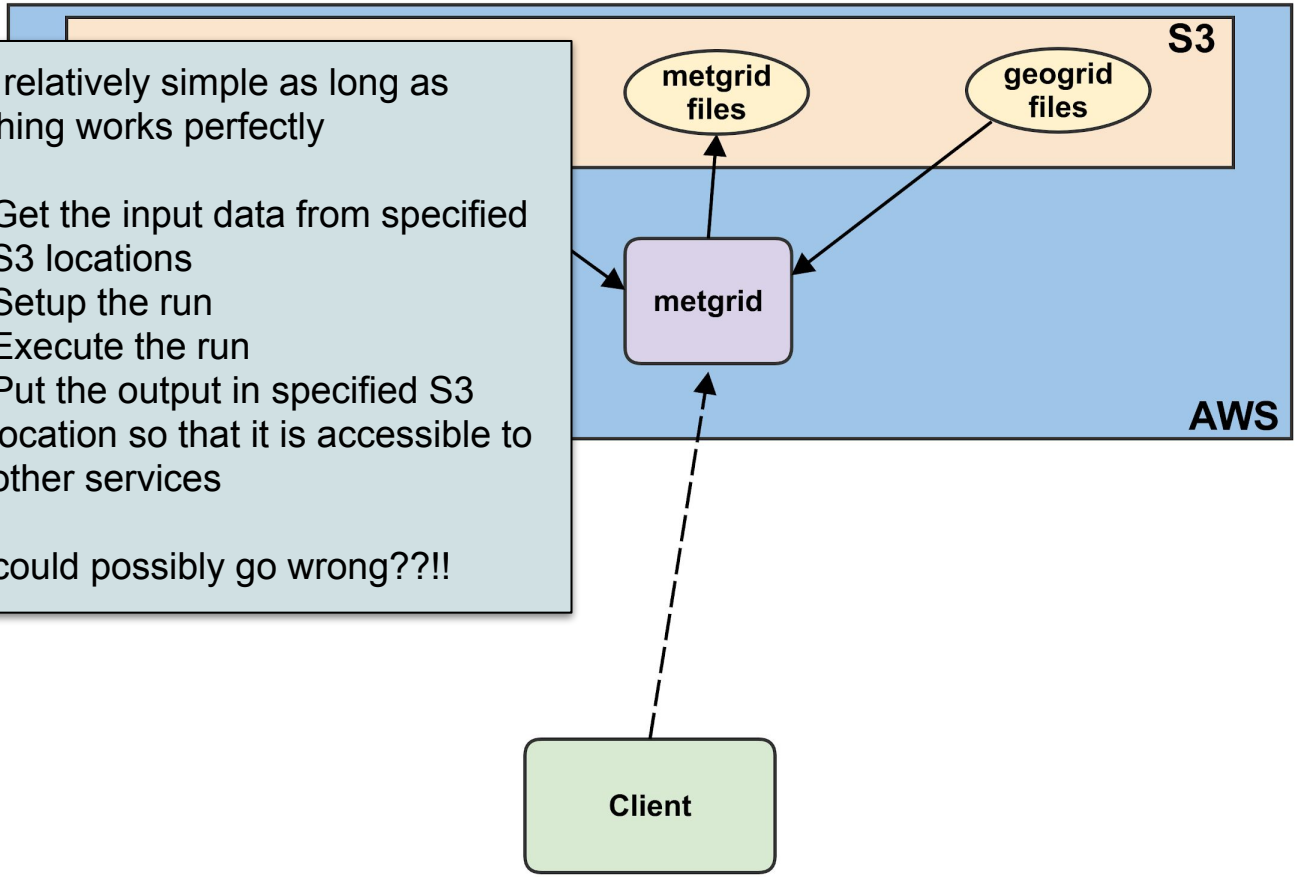


WPS services workflow

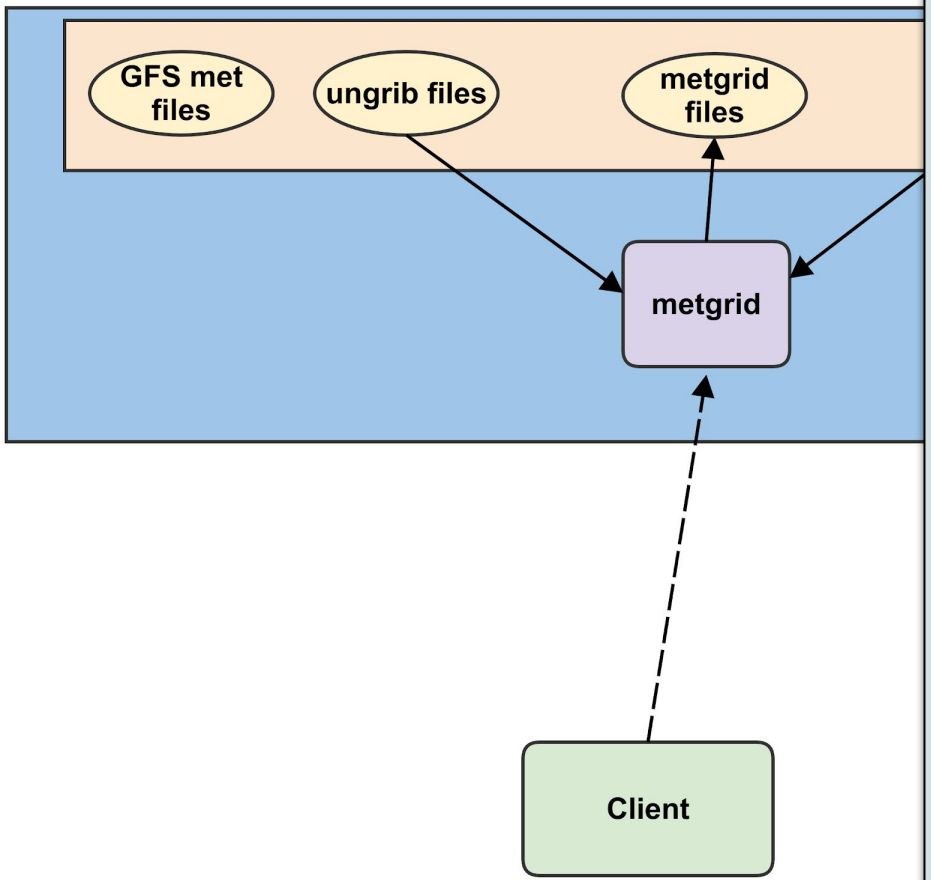
Looks relatively simple as long as everything works perfectly

- Get the input data from specified S3 locations
- Setup the run
- Execute the run
- Put the output in specified S3 location so that it is accessible to other services

What could possibly go wrong??!!



WPS services workflow



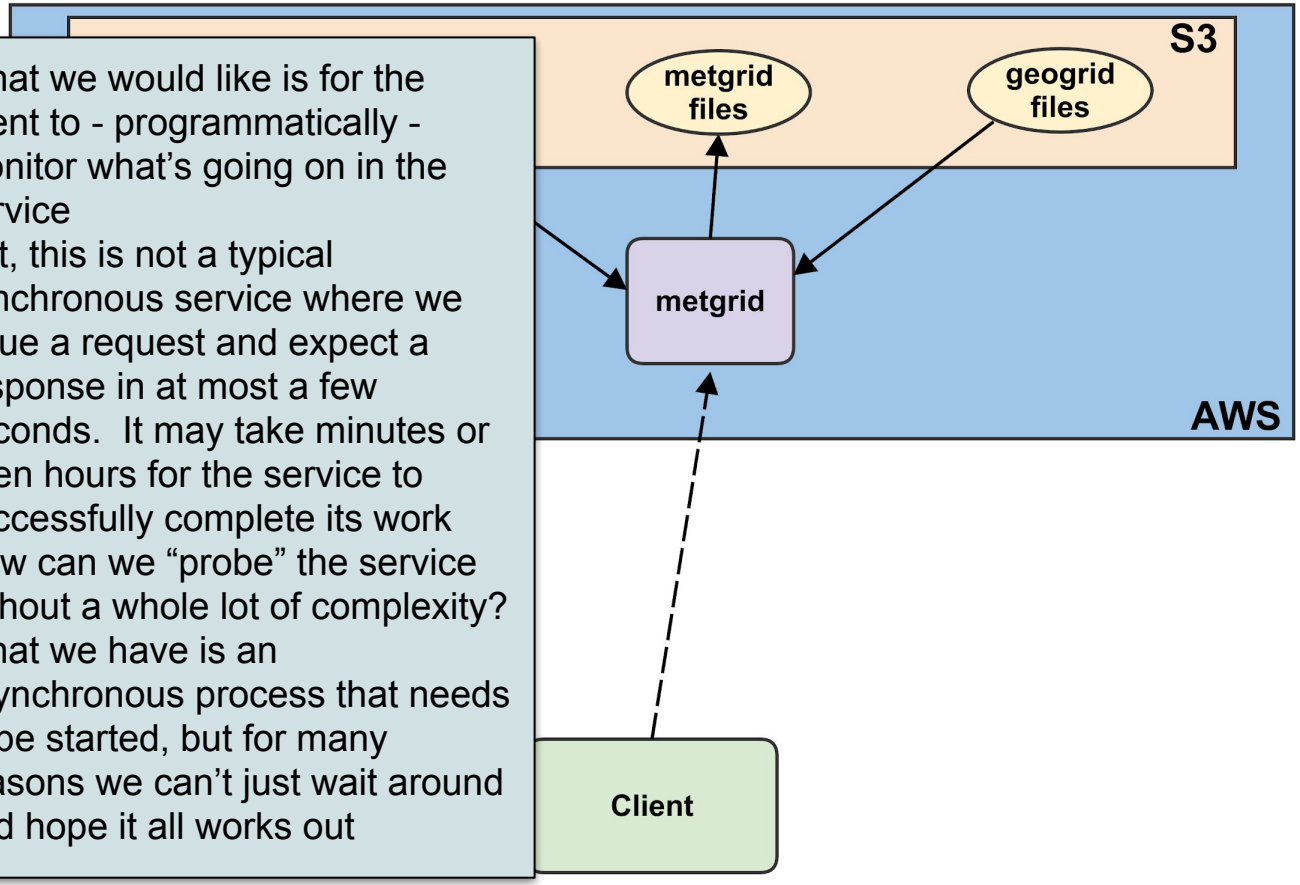
Wellllll, sorry to be so pessimistic, but… we've just launched a service into the cloud with the expectation that our desired output will eventually make it into the S3 location we specified...

- What if the input files aren't where we expect?
- What if copying the input files into the container uses all available disc space?
- What if we didn't allocate enough RAM on our instance?
- What if the *namelist.wps* has a problem?
- How will we know if/when we should give up hope of that output ever appearing in the S3 location?

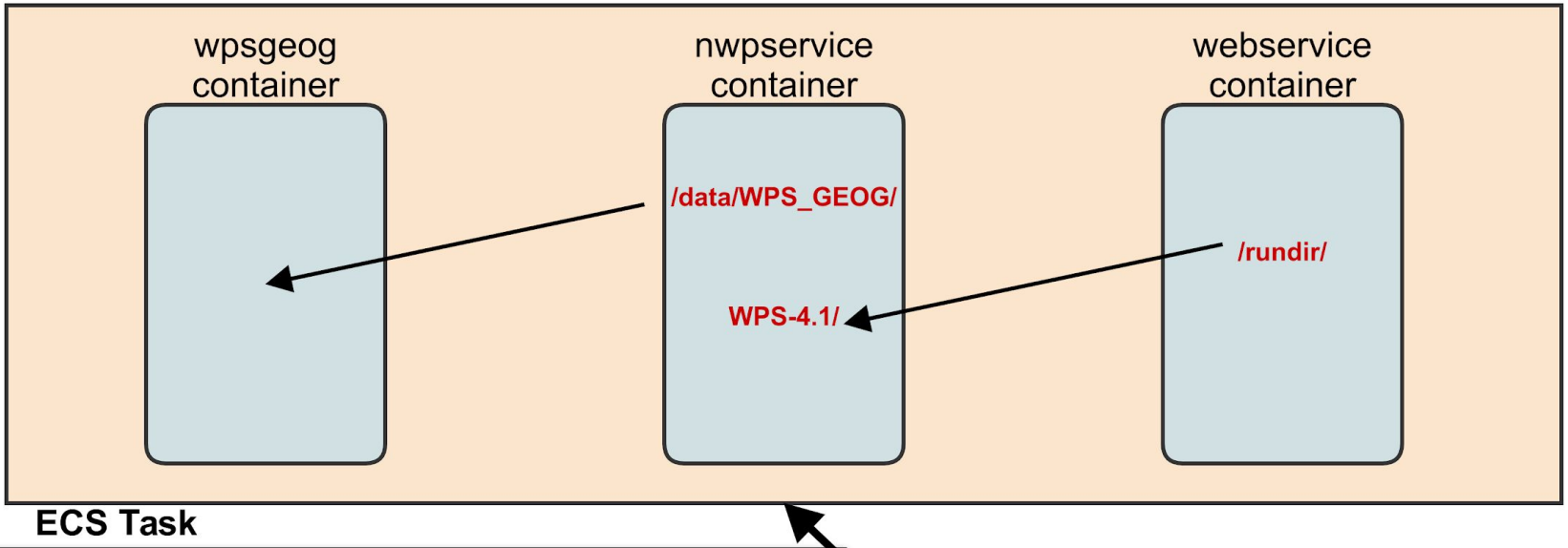
We're only just getting started on the potential problems!

WPS services workflow

- What we would like is for the client to - programmatically - monitor what's going on in the service
- But, this is not a typical synchronous service where we issue a request and expect a response in at most a few seconds. It may take minutes or even hours for the service to successfully complete its work
- How can we "probe" the service without a whole lot of complexity?
- What we have is an asynchronous process that needs to be started, but for many reasons we can't just wait around and hope it all works out



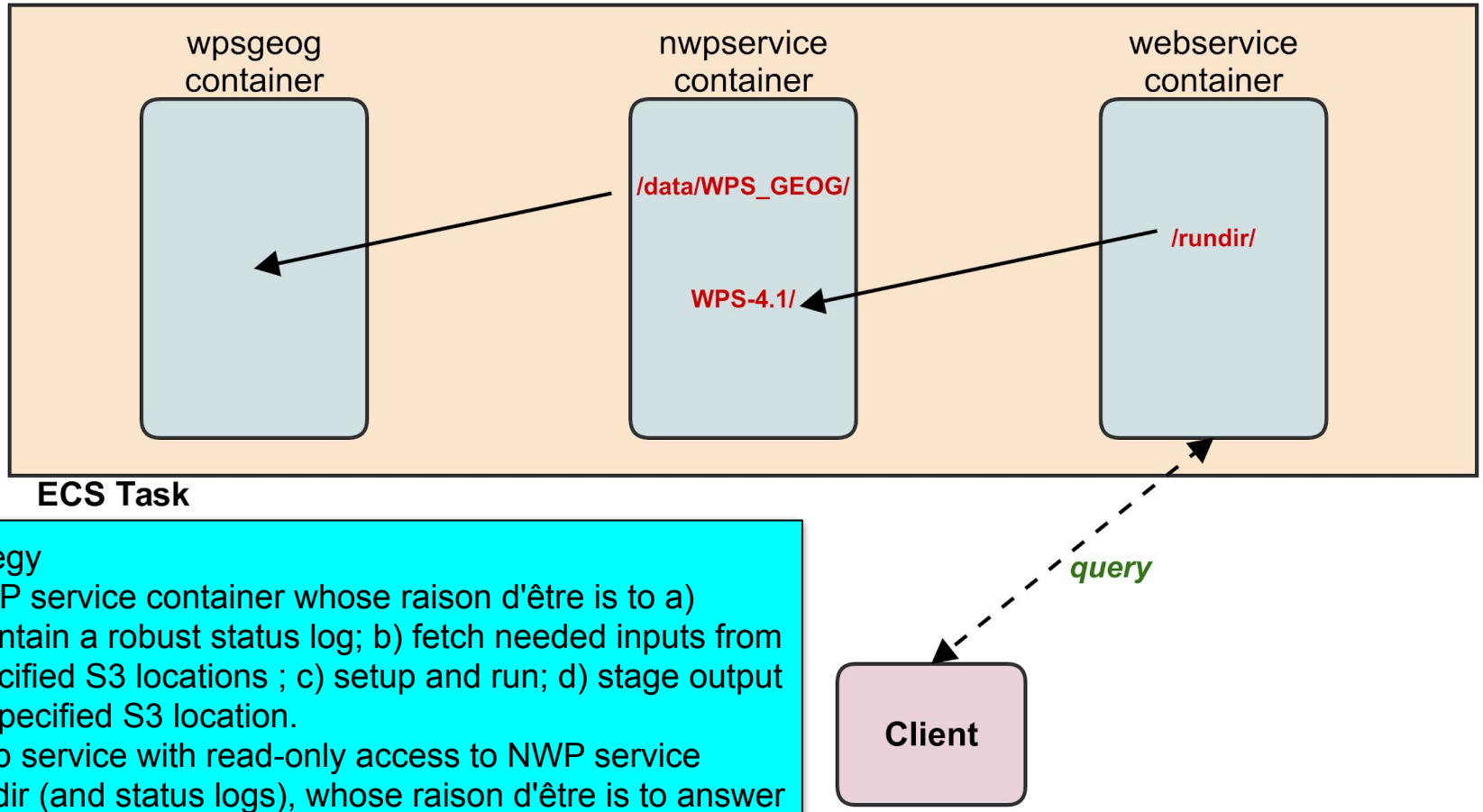
geogrid services ECS task



Our strategy

- NWP service container whose raison d'être is to a) maintain a robust status log; b) fetch needed inputs from specified S3 locations ; c) setup and run; d) stage output to specified S3 location.
- Web service with read-only access to NWP service rundir (and status logs), whose raison d'être is to answer queries about current status of NWP service

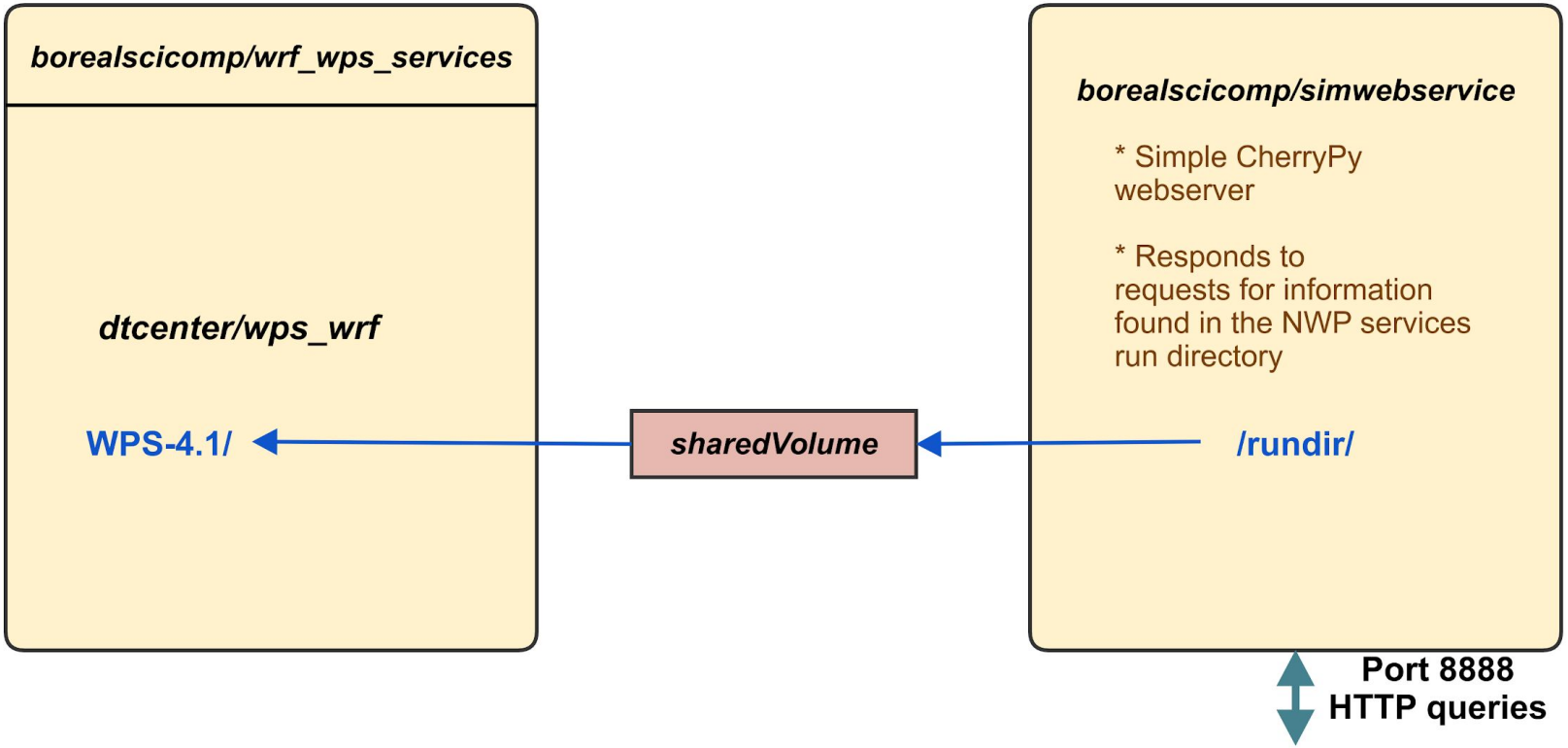
geogrid services ECS task



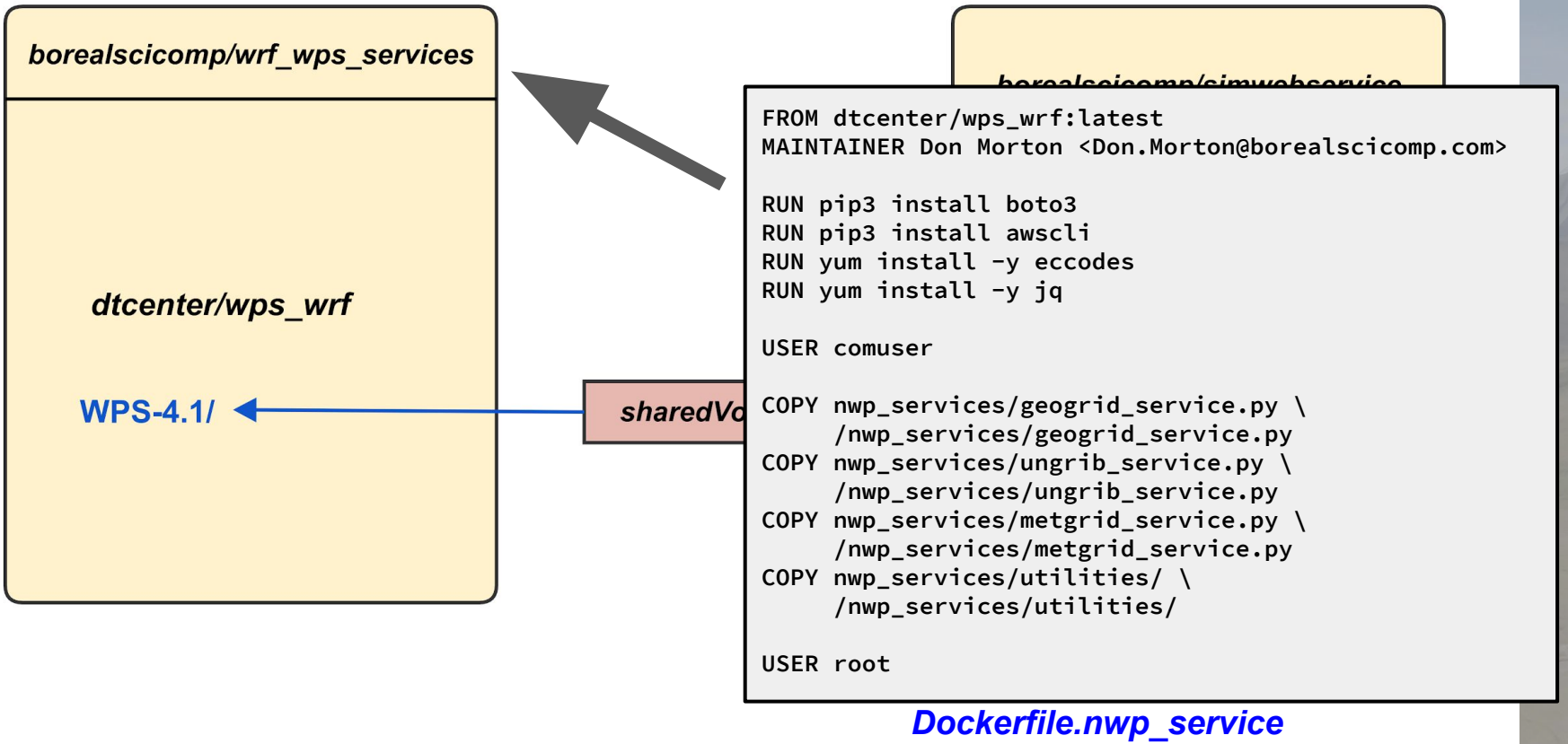
Our strategy

- NWP service container whose raison d'être is to a) maintain a robust status log; b) fetch needed inputs from specified S3 locations ; c) setup and run; d) stage output to specified S3 location.
- Web service with read-only access to NWP service rundir (and status logs), whose raison d'être is to answer queries about current status of NWP service

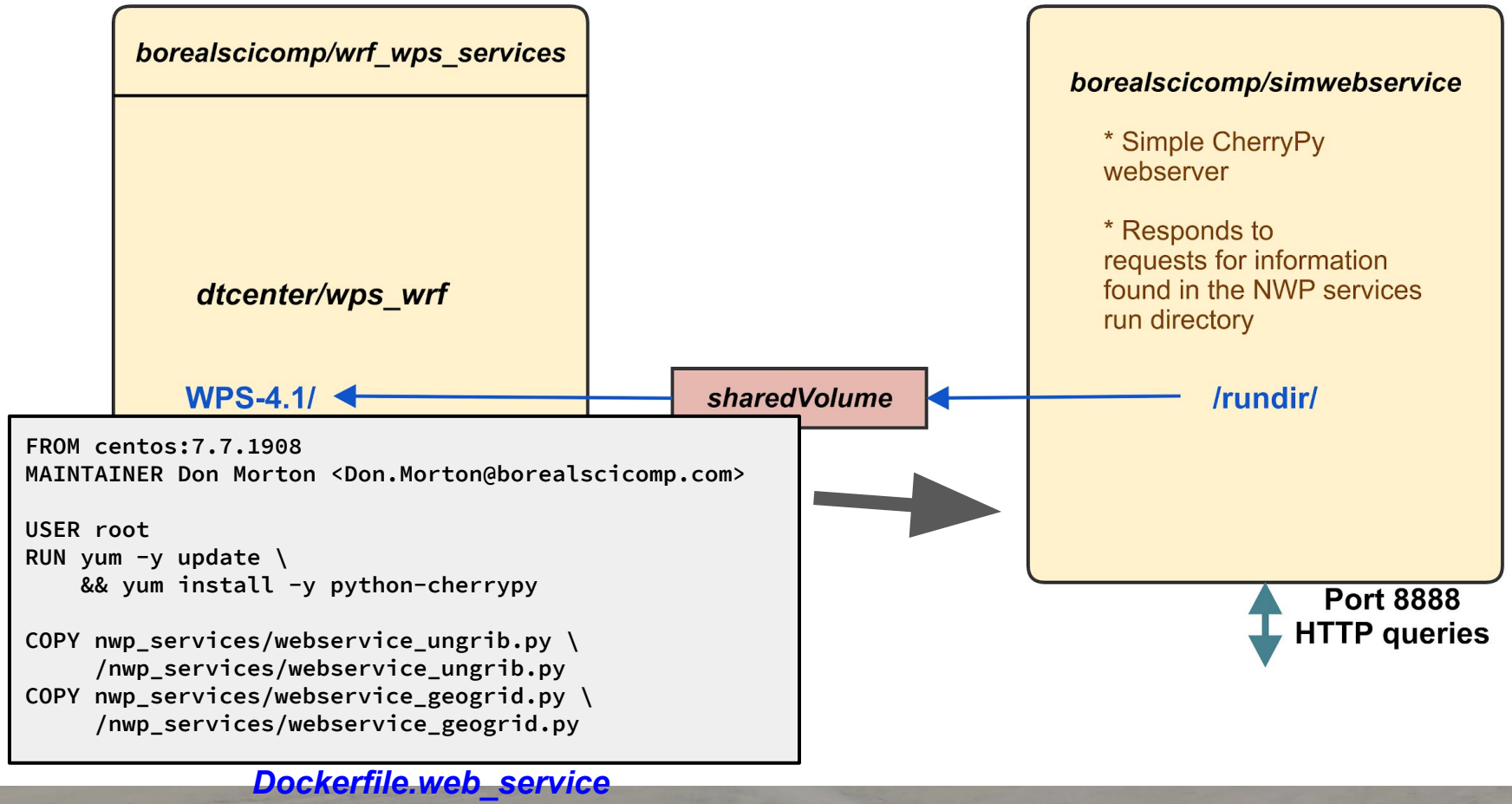
nwp and web services docker components



nwp and web services docker components



nwp and web services docker components




```
FROM centos:7.7.1908
MAINTAINER Don Morton <Don.Morton@borealscicomp.com>

USER root
RUN yum -y update \
    && yum install -y python-cherrypy

COPY nwp_services/webservice_ungrib.py \
     /nwp_services/webservice_ungrib.py
COPY nwp_services/webservice_geogrid.py \
     /nwp_services/webservice_geogrid.py
```

Dockerfile.web_service

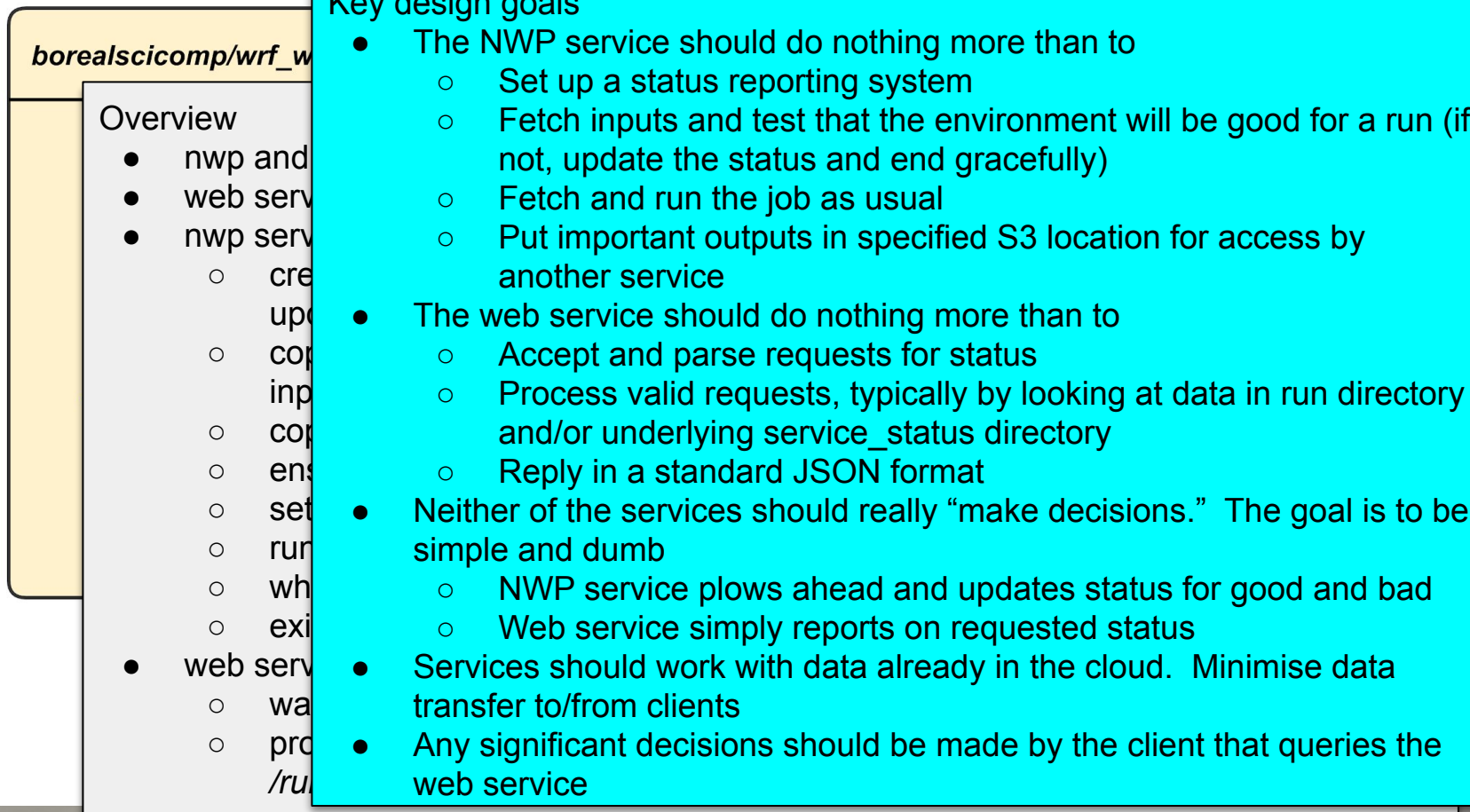
nwp and web services docker components

borealscicomp/wrf_wps_services

Overview

- nwp and web services are launched in separate containers.
- web service has read-only access to nwp service run directory
- nwp service sets up its run directory and runs
  - creates a *service_status/* subdirectory and initialises a *status.json* file, which is updated after various milestones (see following steps)
  - copies in namelist.wps from S3 (uses this to figure out expected filenames of input files)
  - copies in input files from specified S3 location
  - ensures that specified S3 output directory is present and writable
  - sets up whatever needs to be set up in run directory
  - runs it, just like an interactive user would
  - when job is complete, copies outputs (and logs) to specified S3 output location
  - exits
- web service
  - waits for status requests on port 8888
  - processes status requests by looking at data in */rundir/* and */rundir/service_status/*

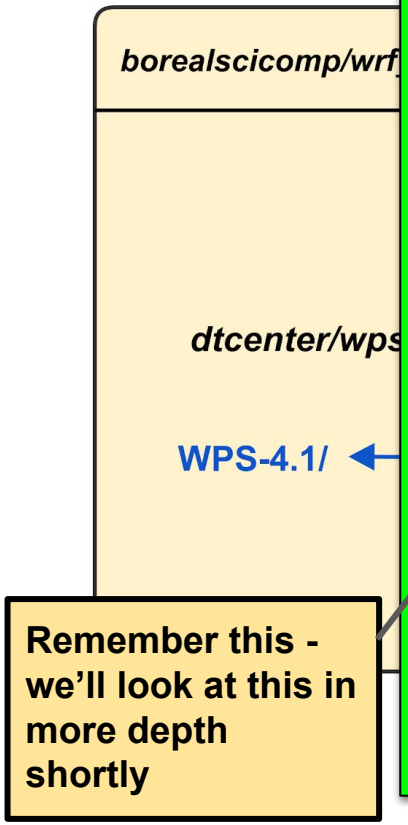
nwp and web services docker components



Key design goals

- The NWP service should do nothing more than to
  - Set up a status reporting system
  - Fetch inputs and test that the environment will be good for a run (if not, update the status and end gracefully)
  - Fetch and run the job as usual
  - Put important outputs in specified S3 location for access by another service
- The web service should do nothing more than to
  - Accept and parse requests for status
  - Process valid requests, typically by looking at data in run directory and/or underlying service_status directory
  - Reply in a standard JSON format
- Neither of the services should really “make decisions.” The goal is to be simple and dumb
  - NWP service plows ahead and updates status for good and bad
  - Web service simply reports on requested status
- Services should work with data already in the cloud. Minimise data transfer to/from clients
- Any significant decisions should be made by the client that queries the web service

Closer look at ungrib service



From the perspective of the client

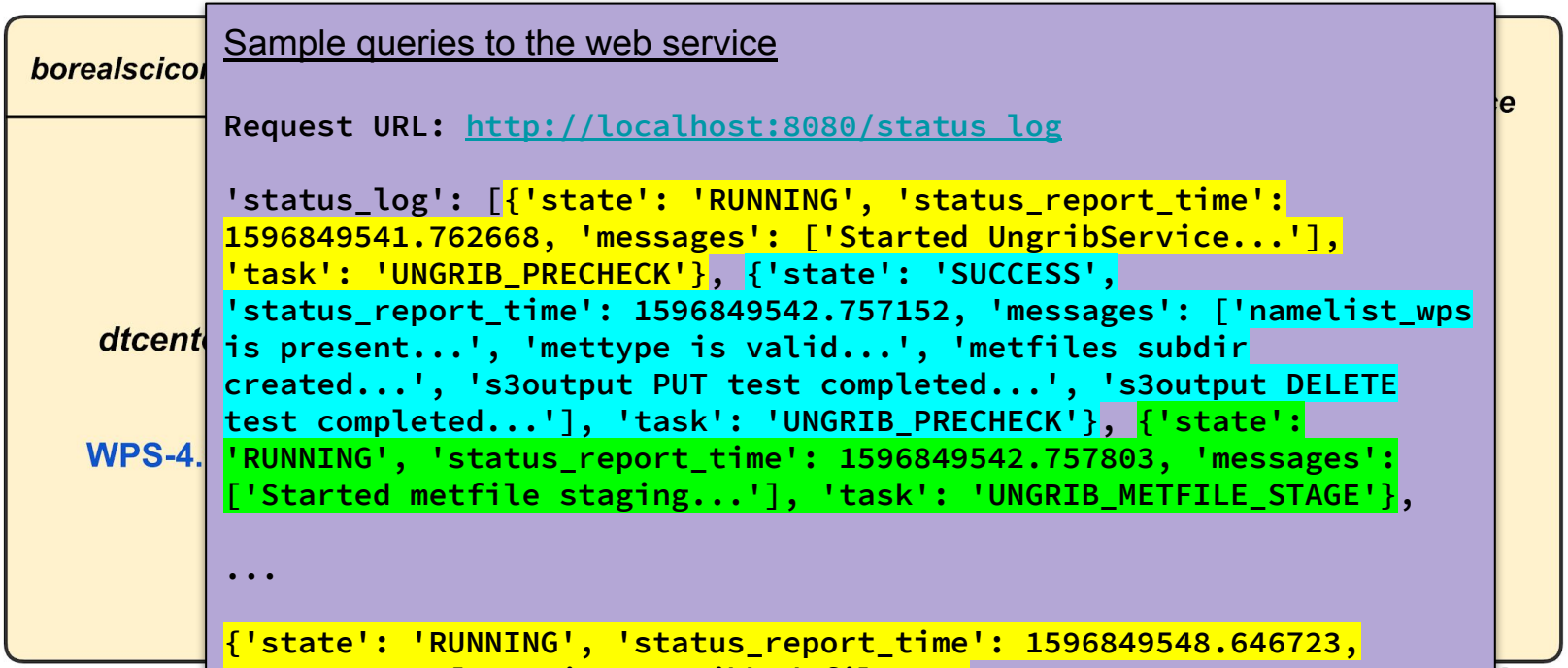
- Understand the S3 locations for input data (often the outputs of previous service calls in the workflow)
- Ensure that desired S3 location for output data is specified and ready
- Put the namelist.wps in an S3 location, accessible by the service
- Start the service (asynchronously)
  - Lots of AWS-specific starting up of VPC's, taskdefs, etc.
  - Run the actual ungrib service in the NWP container

```
ungrib_service.py --s3metbucket <S3 location of GFS data>
                  --mettype 'S3_NOAA_GFS_BDP_PDS_0p50'
                  --s3namelistobj   <namelist.wps S3 location>
                  --s3output  <S3 location for output>
```

- Periodically poll the web service for status updates - these are synchronous operations

Remember this - we'll look at this in more depth shortly

Closer look at ungrib service



Sample queries to the web service

Request URL: http://localhost:8080/status_log


```
'status_log': [{'state': 'RUNNING', 'status_report_time':
1596849541.762668, 'messages': ['Started UngribService...'],
'task': 'UNGRIB_PRECHECK'}, {'state': 'SUCCESS',
'status_report_time': 1596849542.757152, 'messages': ['namelist_wps
is present...', 'mettype is valid...', 'metfiles subdir
created...', 's3output PUT test completed...', 's3output DELETE
test completed...'], 'task': 'UNGRIB_PRECHECK'}, {'state':
'RUNNING', 'status_report_time': 1596849542.757803, 'messages':
['Started metfile staging...'], 'task': 'UNGRIB_METFILE_STAGE'},

...

{'state': 'RUNNING', 'status_report_time': 1596849548.646723,
'messages': ['Staging ungribbed files to
s3://boreal-temporary-01days/ungrib_test_20200806060000_73b11cff-9a
e6-4abc-8a12-fafb51626810'], 'task': 'UNGRIB_OUTPUT_STAGE'},
{'state': 'COMPLETE', 'status_report_time': 1596849601.78797,
'messages': ['Staging ungribbed files to s3 completed BUT NOT
VERIFIED'], 'task': 'UNGRIB_OUTPUT_STAGE'}]}
```

Closer look at ungrib service

Sample queries to the web service

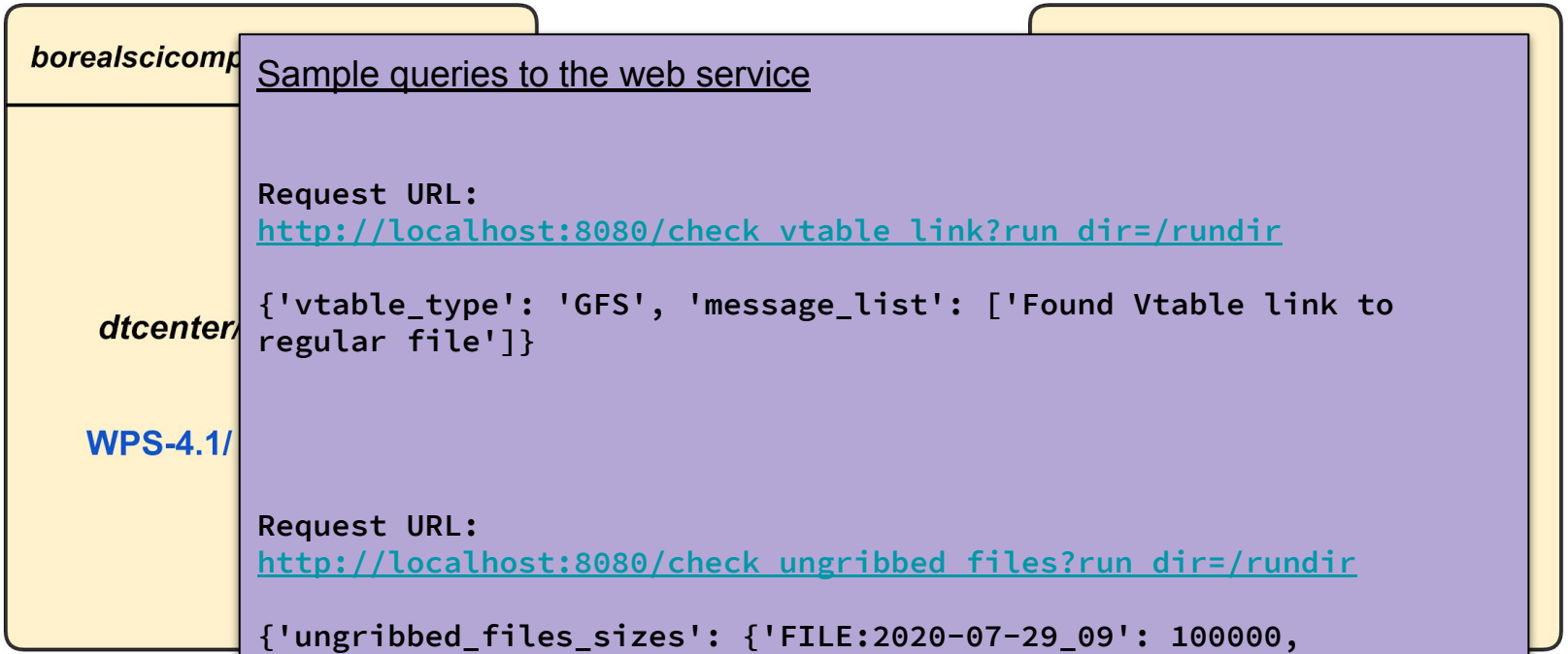
Request URL:
http://localhost:8080/check_staged_metfiles?metfile_dir=/rundir/metfiles

```
{'num_metfiles': 3, 'largest_size_bytes': 100000, 'message_list':
[], 'smallest_size_bytes': 100000}
```

Request URL:
http://localhost:8080/check_staged_metfiles?metfile_dir=/rundir/metfiles_BADDIR

```
{'num_metfiles': 0, 'largest_size_bytes': 0, 'message_list':
['metfile_dir not found: /rundir/metfiles_BADDIR'],
'smallest_size_bytes': 0}
```

Closer look at ungrib service



The very tedious plumbing

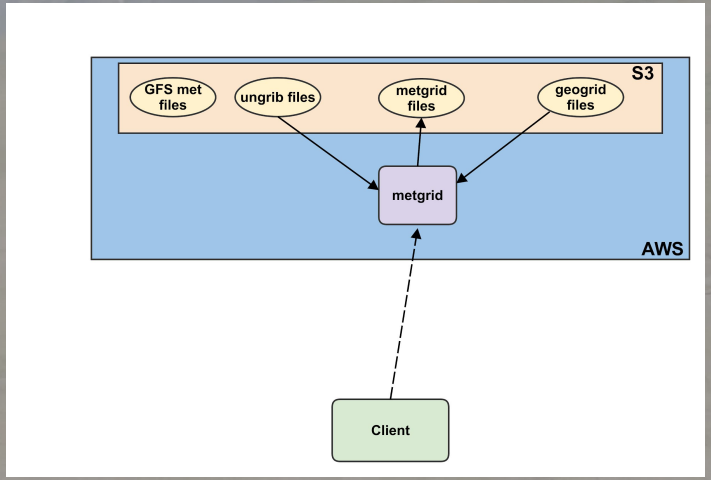
- Implementing the NWP service (on top of the already well-developed DTC NWP containers) isn't very hard
- Implementing the web service interface and coupling it with the NWP container isn't very hard
- It all appears easy-peasy as long as things run without problems
- Therefore, the huge bulk of this work is in hiding away all of the details and cleanly handling the many problems and "gotchas" that may arise

Boreal
Scientific Computing LLC

Under the hood of a “driving client” - Python SDK *boto3*

- AWS credentials
- Insure inputs are in accessible S3 buckets
- Insure output S3 buckets will be available and writable
- VPC setup - includes security group, subnet, routing table, internet gateway
- Set up an ECS cluster name
- Create an EC2 instance for the container task
- Turn it into an ECS instance
- Get the public IP address of the instance (so that we can query the web service)
- Define the ECS task - all container parameters - ports, mount points, memory/cpu requirements etc.
- Start the ECS task (with the NWP and webservice containers) and hope
- When all is done, we need to verify, then successfully tear down the above resources

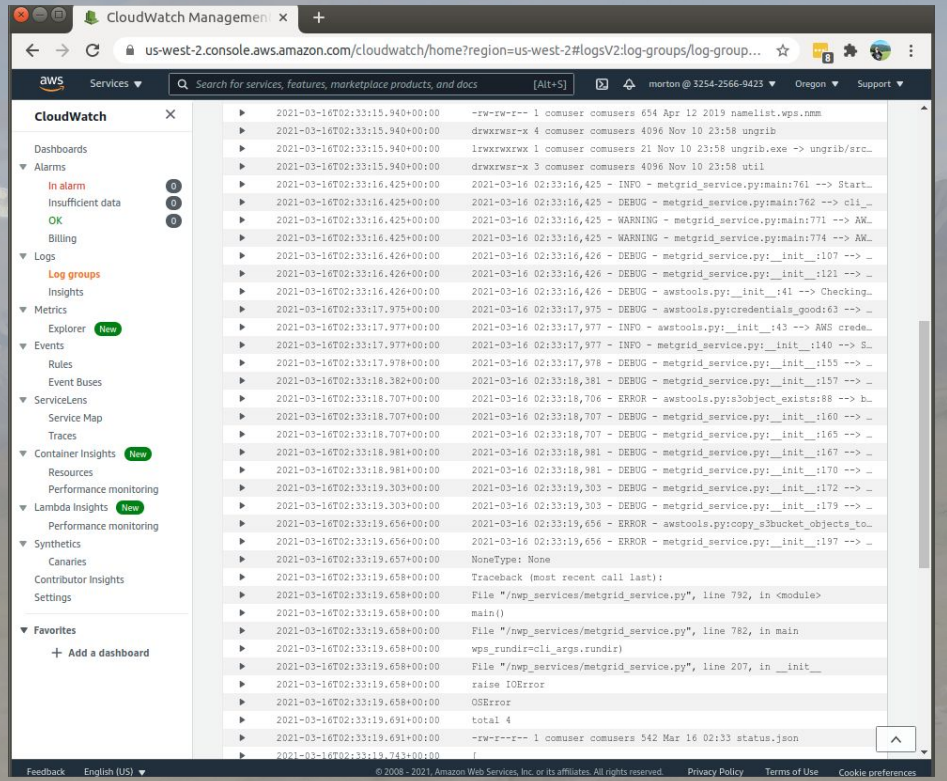
Some of these steps need to be implemented asynchronously - we can’t just plow from one operation to the next without waiting and confirming success



Boreal
Scientific Computing LLC

Development and debugging

- Huge number of degrees of freedom
- Many surprise “gotchas” in a Docker and AWS environment
- Test Driven Development (TDD) is imperative and life-saving
- Debugging in AWS environment can be very slow and frustrating, so as much as possible, initial testing is done in local containers
- Debugging containers in ECS instances requires creative logging and post-execution perusal of CloudWatch logs



Boreal Scientific Computing LLC

Testing

- Use of pytest for driving unit and functional tests
- Very tedious, probably spent more time (and much more code) on building the tests than the services, but it's very important and with all this complexity, has kept me grounded
- Last count 102 tests and climbing!
- Many of the tests are done on local containers (sometimes accessing S3 objects remotely), and many include full AWS ECS deployment

Boreal
Scientific Computing LLC

```
[dtcuser@localhost tests]$ pytest
================================= test session starts ==================================
platform linux -- Python 3.6.8, pytest-5.4.3, py-1.9.0, pluggy-0.13.1
rootdir: /home/dtcuser/git/dockercloudwrf/wps_wrf/tests
collected 102 items

test_services/test_geogrid_system_after_aws_run.py .....                          [  4%]
test_services/test_geogrid_system_after_local_run.py ....                         [  8%]
test_services/test_nwp_standalone_geogrid_service.py .                            [  9%]
test_services/test_nwp_standalone_geogrid_service_after_aws_run.py ..             [ 11%]
test_services/test_nwp_standalone_metgrid_service_basictest001.py .               [ 12%]
test_services/test_nwp_standalone_ungrib_service_after_aws_run_basictest001.py .  [ 13%]
test_services/test_nwp_standalone_ungrib_service_after_aws_run_with_realtime_aws_gfs.py . [ 14%]
                                                                                  [ 14%]
test_services/test_nwp_standalone_ungrib_service_basictest001.py .                [ 15%]
test_services/test_nwp_standalone_ungrib_service_with_realtime_aws_gfs.py .       [ 16%]
test_services/test_standalone_webservice_geogrid.py .........                     [ 25%]
test_services/test_standalone_webservice_ungrib.py ............                  [ 37%]
test_services/test_ungrib_system_after_aws_run_basictest001.py ........           [ 45%]
test_services/test_ungrib_system_after_aws_run_with_realtime_aws_gfs.py ........  [ 52%]
test_services/test_ungrib_system_after_local_run_basictest001.py .......          [ 59%]
test_services/test_ungrib_system_after_local_run_with_realtime_aws_gfs.py .......  [ 66%]
test_services_utilities/func/test_awstools.py ..................                  [ 84%]
test_services_utilities/unit/test_namelist_wps.py ................                [100%]

============================== 102 passed in 1802.98s (0:30:02) ==============================
```

Next steps

- WPS backend client prototype almost done
- Pause on the low-level development and build Python-based API examples for driving the WPS components in real-world scenarios
  - Preliminary user test
  - Documentation
- Move on to implementation of additional WRF and NWP services
- Assess where we're at and what it would take to put all of this behind RESTful APIs for micro(haha) services

Boreal
Scientific Computing LLC

Conclusions, summary

- This needs to be done. It's hard and tedious, and I don't know of an easy way to do it. But, somebody needs to do it
- The "grid" paradigm has been promoted for well over twenty years
- Building the RESTful interface is the ultimate aim. From there we can build all kinds of interfaces for all kinds of uses
- Unlike the early "Grid" days, the pieces are all out there, available to anybody. The challenge is in building the underlying plumbing and the overlying abstractions

Boreal
Scientific Computing LLC