Paul Maravelias
4/9/2019

TempoQuest AceCAST^{TM} On-Demand Web Application

Initial specification for emulator of DI and JMT APIs on created AWS compute instances for development and testing of automated Job Execution with browser web application.

This document explains emulator usage only. For the actual implementation steps for the DI and JMT during Job Execution, refer to Paul's "April2019JobExecutionAPISteps.pdf" (hereinafter, "specification") emailed on 4/1/19.

1. DI SUBMIT EMULATOR

a. Comment out the real endpoint (which is http://**$**DYNAMICALLYOBTAINEDIPADDRESS:5226/api/v1.0/di/**q**) and use static emulator link to POST the DI message:

```
http://54.173.193.93/di_emulator_q.php?testId=$INSERTUNIQUESESSIONID
```

b. Notice that for the emulator you have to generate a unique session id, probably needing to be saved in the DB, for use later in the DI POLLING phase. This is so that we can have the emulator doing multiple jobs at once (need to keep track of which jobs are which). Maybe there's already a convenient "model job ID" variable in the DB you can use.

c. The emulator does NO VALIDATION WHATSOEVER ON THE POSTED DI MESSAGE, only to make sure it exists in the HTTP call. Unlike the real DI, returned download links will have NOTHING to do with the actual links used by the particular model config. This should have NO IMPACT on the web application when we switch to real DI, because the specification for this step already requires that we interpret the JSON response links here and use them in the subsequent polling calls.

d. The response from this emulator endpoint will ALWAYS be the following for test purposes, which still needs to be dynamically parsed using JSON decode (see specification) to make and memorize the list of being-downloaded URLs:

{"responseCode":0,"responseMsg":"Download initiated","statuses":[{"url":"http://www.ftp.ncep.noaa.gov/data/nccf/com/hrrr/prod/hrrr.20190311/conus/hrrr.t00z.wrfprsf00.grib2","state":"Downloading"},{"url":"http://www.ftp.ncep.noaa.gov/data/nccf/com/hrrr/prod/hrrr.20190311/conus/hrrr.t01z.wrfprsf

00.grib2","state":"Downloading"},{"url":"http://www.ftp.ncep.noaa.gov/data/nccf/com/hrrr/prod/hrrr.20190311/conus/hrrr.t02z.wrfprsf00.grib2","state":"Downloading"}]}
NOTE: There is also no validation on calling DI SUBMIT twice. When DI SUBMIT is called, it will simply reset the whole DI progress for that particular unique ID. It should be very easy in the server-side code to make sure DI SUBMIT is just called once, and then we progress to DI POLLING after.

2. DI STATUS POLLING EMULATOR

a. Comment out the real endpoint (which is http://**$**DYNAMICALLYOBTAINEDIPADDRESS:5226/api/v1.0/di/**status**) and use static emulator link to POST the all the returned DI urls to obtain download statuses:

   `http://54.173.193.93/di_emulator_status.php?testId`**`=$UNIQUESESSIONID`**

b. Use the unique session ID we generated on DI SUBMIT (emulator days only, unnecessary in real DI)

c. The response will first be:

   {"responseCode":0,"responseMsg":"Download initiated","statuses":[{"url":"http://www.ftp.ncep.noaa.gov/data/nccf/com/hrrr/prod/hrrr.20190311/conus/hrrr.t00z.wrfprsf00.grib2","state":"**Downloading**"},{"url":"http://www.ftp.ncep.noaa.gov/data/nccf/com/hrrr/prod/hrrr.20190311/conus/hrrr.t01z.wrfprsf00.grib2","state":"**Downloading**"},{"url":"http://www.ftp.ncep.noaa.gov/data/nccf/com/hrrr/prod/hrrr.20190311/conus/hrrr.t02z.wrfprsf00.grib2","state":"**Downloading**"}]}

d. After ten seconds, the first file will go to “state” = “Complete”. After another ten seconds, the second file. And after another ten seconds, the third. At that point all files’ “state” will != “Downloading”, which is the logical test to continue to JMT (see specification). Obviously these response messages need to be dynamically parsed with JSON decode and then we check the “state” field for each.

3. JMT SUBMIT EMULATOR

a. Comment out the real endpoint (which is http://**$**DYNAMICALLYOBTAINEDIPADDRESS:5226/api/v1.0/jmt/job/**create**) and use static emulator link to POST the JMT message:

`http://54.173.193.93/jmt_emulator_create.php`

b. Just like the DI emulator endpoint, this does NO VALIDATION on the contents and format of the huge JMT JSON message other than making sure it's POSTed.[1]

c. The response will be as follows:

{"responseCode":0,"responseMsg":"WpsRunning","detailedMsg":[],"userId":1, "jobId":**$GENERATEDUNIQUEJOBIDINTEGER**,"jobResponseCode":3}

Obviously, as specified in the specification, we need to parse the "jobId" integer and store it for the following JMT status calls.

4. JMT STATUS EMULATOR

a. Comment out the real endpoint (which is http://**$**DYNAMICALLYOBTAINEDIPADDRESS:5226/api/v1.0/jmt/job/**status/$JOBID**) and use static emulator link to GET the JMT status message (nothing to POST here):

`http://54.173.193.93/jmt_emulator_status.php?jobId`**`=$JOBID`**

b. Obviously replace "$JOBID" with the stored jobId returned from the JMT submit.

c. The response will first be:

{"responseCode":0,"responseMsg":"**jmtStage1**","detailedMsg":[],"userId":1,"jobId":**$**JOBID,"jobResponseCode":3}

d. Then, after 10 seconds,

{"responseCode":0,"responseMsg":"**jmtStage2**","detailedMsg":[],"userId":1,"jobId":**$**JOBID,"jobResponseCode":3}

e. And so forth, until finally:

[1] We already did an initial validation of the JMT JSON generation (even though I did recently notice and mention on WhatsApp the vertical levels problem – selection in model config UI does not change vertical levels at all in JMT JSON or web page UI in terms of updating the number of grid points – is that fixed?)

{"responseCode":0,"responseMsg":"**Complete**","detailedMsg":[],"userId":1,"jobId":**$**JOBID,"jobResponseCode":3}

f. To repeat what's already stated in the specification, for every status message, we check for a different/changed "responseMsg" and populate/update the Job Execution UI web page accordingly on the front end. Just display the textual contents of "responseMsg" on the web page – obviously in the real JMT it's not "jmtStageX" but the actual names of the actual WRF steps. We also obviously perform the logical test for "responseMsg = 'Complete'", at which point we are DONE (update UI for completion of JMT and shut-down/delete CMT).