KUBE Documentation

Introduction:

KUBE is a benchmarking and testing framework rather than a benchmarking 'suite', as it is not our objective to gather and define a bunch of applications or software tests along with proper datasets. This latter is the user's responsibility. As a benchmarking and testing framework, our goal is to create a means to drive, centralize, control and organize the benchmarking and testing process in a consistent, flexible and easy to use way. In this manner any benchmark may be added and managed using KUBE.

KUBE is driven by a configuration file that contains rules to be applied in the different stages of the benchmarking/testing process. The configuration file may be specified as an argument, otherwise the default "etc/kube.yaml" file will be used. The user may also develop custom configuration files (for example if you have different machines and environments) and choose to use them as needed.

Directory structure:

As a means to understand KUBE's process and configuration file, we first advise understanding the directory structure of the framework.

The directory structure is as follows:

```
|--- bench/   ### Contains categorized benchmarks
|       |--- apps/
|       |--- filesystems/
|       |--- networks/
|       |--- synthetics/
|--- bin/   ### The main scripts live here
|--- doc/
|--- etc/   ### Contains the configuration file(s) and the templates used by KUBE
|       |--- kube.yaml -> noor.yaml
|       |--- noor.yaml
|       |--- run.lsf.in
|--- lib/
|       |--- kube  ### contains the core engine scripts
|       |--- PyYAML-3.10
|       |--- yaml -> PyYAML-3.10/lib/yaml/
|
|--- data/
       |--- results/   ### Contains analysis for each run with the same structure as the 'runs' dir below
       |       |--- apps/
       |       |--- filesystems/
       |       |--- networks/
       |       |--- synthetics/
       |---  runs/   ### Contains the actual execution directories for every run
       |       |--- apps/
       |       |--- filesystems/
```

```
       |       |--- networks/
       |       |--- synthetics/
       |---  rrd/   ### Contains the RRD databases for the metrics in every run
               |--- apps/
               |--- filesystems/
               |--- networks/
               |--- synthetics/
```

Inside *bench/apps* there is a list of application names exactly as they appear in the configuration file. Each application should have at least one dataset, but may contain more than one. Inside an app dir, the system should be able to find the files **<dataset_name>.tgz** being **<dataset_name>** the name of the dataset as it appears in the configuration file.

The rest of items: filesystems, networks and synthetic are very similar but different to the apps because while each app expects to find one or several self contained datasets as a .tgz files; the rest of items expect to find one or several directory entries to each "dataset" . Since filesystems, networks and synthetic can be understood as "synthesized" benchmark (or "artificial", "ad-hoc", ...) they are not likely to contain a dataset in an application sense, but rather a full dir structure with binaries and dependencies that need to be compiled and called with certain arguments. For these reasons, these items have an additional tag in the configuration file: "dependencies". This can be a comma separated list of things that is needed to be copied to the run directory in order to be able to create a self contained dir to run it. Of course it is allowed to use wildcards here, so you can write something like:

```
dependencies: bin/* , lib/*
```

if you need to copy all the contents of the bin dir along with the libs

There are more details regarding the configuration file below.

Usage:

The main script is a Python script named 'kube'. It's been written in python2.7 and can be called with different argument each defining a specific action: **view**,**run**,**refine** and **probe**. And each action may be controlled with options. The following is the help message shown by kube:

```
$ kube -h
usage:
The accepted ways to run KUBE are:
---------------------------------
kube [ {-h,--help} ]
kube {-v,--version}
kube <COMMAND> <OPTIONS>

Some <OPTIONS> are specific to some <COMMANDS>:
-----------------------------------------------
kube clean   [-d {runs,results}] [-a APPS] [-n NETS] [-f FILESYS] [-s SYNTHS] [--since SINCE] [--to TO]
kube view    [-a APPS] [-n NETS] [-f FILESYS] [-s SYNTHS]
kube run     [-a APPS] [-n NETS] [-f FILESYS] [-s SYNTHS] [--log FILE]
kube refine  [-a APPS] [-n NETS] [-f FILESYS] [-s SYNTHS] [--since SINCE] [--to TO] [--log FILE] [--rrd] [--force]
kube probe   -t DIR [-p] [-m METRIC_NAMES] [--since SINCE] [--to TO]
kube probe   -b BASE_DIR [-p] [--target DIR] [-m METRIC_NAMES] { [--since SINCE] [--to TO] , [--at DATE] } }

Commands allowed in KUBE:
-------------------------
       {clean,view,run,refine,probe}

optional arguments:
-------------------
 -h, --help            show this help message and exit
 -v, --version         show program's version number and exit
 --since SINCE         Option to set the initial time when needed. Default is the
                       first registered date
 --to TO               Option to set the end time when needed. Default is 'Now'
 --at DATE             Option to select a specific date when needed
```

```
 --log FILE            Print out log information to FILE
 --rrd                 Used along with the command 'refine' to enable RRD
                       integration. Each metric computed is stored in a RRD
                       database
 --force               The 'refine' stage is only done if the configuration section
                       for the current dataset has changed. You may want to force
                       the 'refine' stage for some reason,ie: create rrd database
                       for specific dates that are already refined
 -d {run,results}      Option to specify which directory is to be cleaned
                       when the 'clean' command is used
 -a APPS               Option used to select which apps benchmark from the
                       config file will be processed. This option may accept
                       a comma-separated list if multiple apps need to be
                       selected. Use 'all' to select everything. It is
                       allowed in the following commands: 'run', 'view',
                       'refine'
 -n NETS               Option used to select which network benchmark from the
                       config file will be processed. This option may accept
                       a comma-separated list if multiple apps need to be
                       selected. Use 'all' to select everything. It is
                       allowed in the following commands: 'run', 'view',
                       'refine'
 -f FILESYS            Option used to select which filesystem benchmark from
                       the config file will be processed. This option may
                       accept a comma-separated list if multiple apps need to
                       be selected. Use 'all' to select everything. It is
                       allowed in the following commands: 'run', 'view',
                       'refine'
 -s SYNTHS             Option used to select which synthetic benchmark from
                       the config file will be processed. This option may
                       accept a comma-separated list if multiple apps need to
                       be selected. Use 'all' to select everything. It is
                       allowed in the following commands: 'run', 'view',
                       'refine'
 -t DIR                Option to be used with the 'probe' command to select the
                       time analysis. The value expected is the path to the
                       dir containing the post-processed runs
 -m METRIC_NAMES       For the time and metrics analysis one or more metrics can be
                       selected using this option. A comma-separated list is
                       used for multiple metrics selection
 -b BASE_DIR           Option to be used with the 'probe' command to select the
                       box plot metrics graph analysis
 --target DIR          For the metrics graph analysis you may want to specify a dir
                       path which contains one or many runs information to compare
                       against the base dir provided with -k option. This target dir
                       could be either a dir containing a single run for a specific
                       date or a dir containing several runs for various dates or a
                       dataset dir.
 -p                    When used with -t or -b options, it means that the output will
                       be printed out to stdout instead of creating a graph
```

Let's go over every action and its options:

Clean

The clean action is used to remove some contents in order, for example, to free some space on disk or eliminate unnecessary information.

The syntax is:

```
$ kube clean   [-d {runs,results}] [-a APPS] [-n NETS] [-f FILESYS] [-s SYNTHS] [--since SINCE] [--to TO]
```

Without arguments, this action will remove everything stored in 'runs' and 'results' wherever they are, according to what is given in the configuration file. But you may choose to delete only the runs dir (-d runs) or the results dir (-d results). Or even more you can just provide a comma separated list of APPS, NETS, FILESYS or SYNTHS to remove. Additionally you can select a date range to remove using SINCE and TO options. By default SINCE is always the first record and TO is always 'now'.

View

This is useful to check the current configuration for the whole benchmark, or a specific item. No more needed to say :)

```
$ kube view    [-a APPS] [-n NETS] [-f FILESYS] [-s SYNTHS]
```

Run

Calling 'run' without arguments will run all the configured benchmarks, or a specific item or a list of items. Normally the info is printed out to the standard output but if the user selects the --log option, Then It will be stored in the specified file. This could be useful, for example when running kube in a cron and want to log the info.

```
$ kube run     [-a APPS] [-n NETS] [-f FILESYS] [-s SYNTHS] [--log FILE]
```

This stage creates a run dir for each item with a timestamp. If for some reason the benchmark can't be submitted or run, the directory is marked as 'INVALID'.

Refine

'Refine' is a core step since, according to kube's philosophy, runs and results are stored separately. It means that the refine stage is necessary to post-process the runs and keep only the necessary data along with the metrics defined in the configuration file. Runs dir can then be safely removed when not needed or kept for specific users needs. The **lifespan** entry in the configuration file defines when the runs dir will be removed and it is checked every time the refine stage is run. Every run whose metrics cannot be computed is considered 'INVALID'.

```
$ kube refine  [-a APPS] [-n NETS] [-f FILESYS] [-s SYNTHS] [--since SINCE] [--to TO] [--log FILE] [--rrd] [--force]
```

Again, as in the 'run' stage, you can refine subsets of the benchmark using the options -a, -n, -f and -s and use time filtering with SINCE and TO and log to file with --log option. But here we have two more additional options: --rrd and --force

The option --rrd enables the RRDTool integration and the computed metrics are stored in a database. For those who are familiar with RRDTool, this is the configuration used to store all the metrics:

RRD step: 6h
DS name: name of the metric limited to 19 characters
Heartbeat: 48 hours
RRA 1: keep one sample every 6 hour for 1 month
RRA 2: Weekly average for one year
RRA 3: Monthly average for 2 years

From the above you can see that we chose 6h as step. The reason for that is that we don't believe you will be running the benchmark every 5 minutes and may be 6h gives a good resolution. Besides, the heartbeat is set to 48 hours which means that you may not run the benchmark for 2 days and still get valid data.

The --force option can be used every time the user wants to force the refine stage to be done. 'Refine' stage is only done in case the system finds a new run which has not been refined yet or if the section in the configuration file that affects the current datasets changes. But for some reason the user may want to force to do the refine stage.

Probe

This action is used to analyze and interpret the data, normally through a visual representation. The two types of representations allowed are: metrics' time evolution and metrics comparison using box plots.

Let's start with the time analysis. With the 'probe' command, the option to activate the time analysis is '-t'

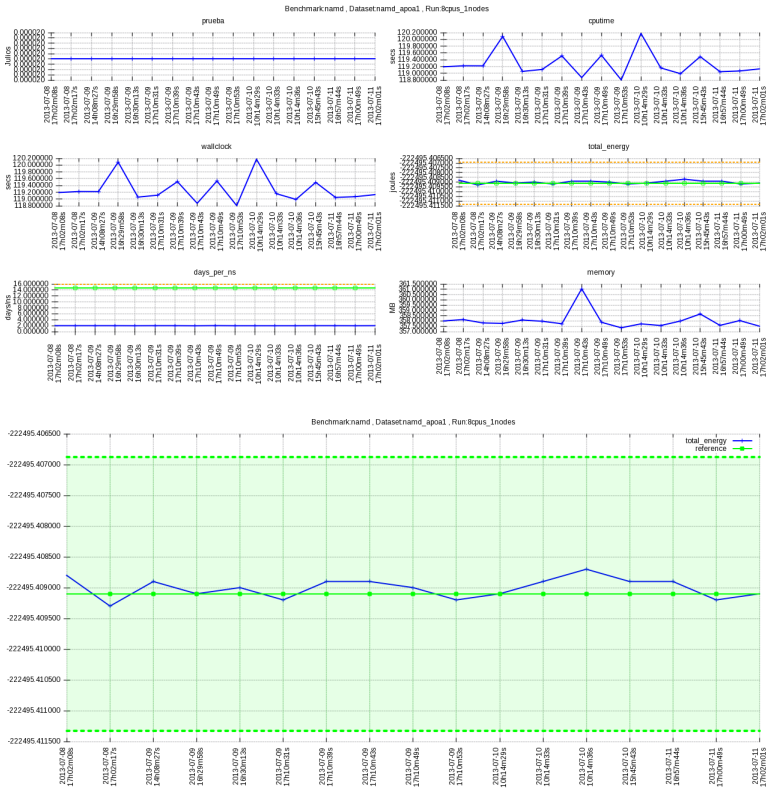
```
$ kube probe -t DIR [-p] [-m METRIC_NAMES] [--since SINCE] [--to TO]
```

DIR must contain a list of directories named with the Kube's timestamp format. For example, inside DIR you can find subdirs named like: **2013-07-08T17h02m17s**. The system will search this directories and figure out the metric values to display. User may also want to define just a range of dates to display (a sort of Zoom In effect) using SINCE and TO. The format is the same. For example if you set **--since 2013-07-08T17h** the system will display all the values from results generated after July 8th,2013 at 5PM until 'now'. Additionally with -m, users can select a list o metrics to display separated by comas. In case the user does not want to display the values but just print them in the screen, they can use the -p option. The columns displayed are:

1.- Timestamp
2.- Value
3.- Reference value
4.- Upper valid limit
5.- Lower valid limit

The last three are only displayed if they exists.

Here are some examples:



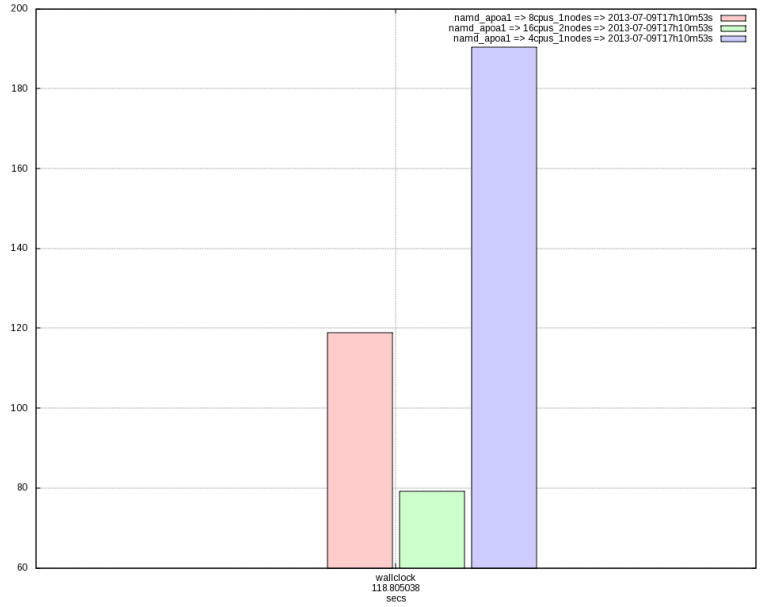
The second way to visualize the metrics is using the box plot diagram.

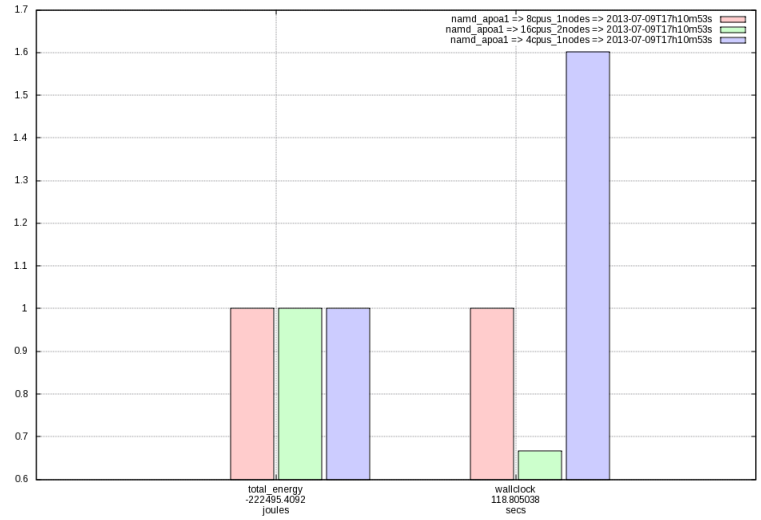
```
$ kube probe   -b BASE_DIR [-p] [--target DIR] [-m METRIC_NAMES] { [--since SINCE] [--to TO] , [--at DATE] } }
```

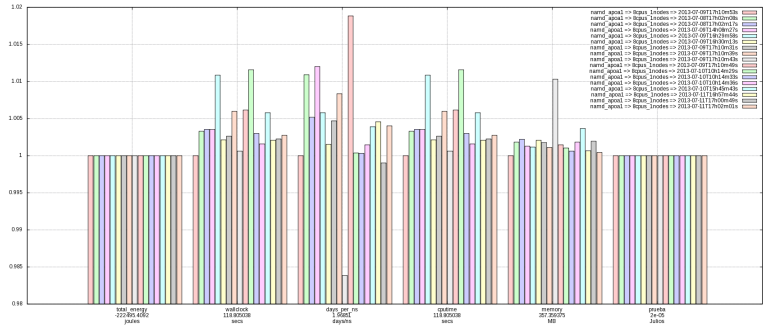
Here BASE_DIR is the path to the final refined directory, the one with the timestamp name. In this directory Kube will read all the metrics info and will display the information in a box plot or to the screen if the -p option is used. The real added value of the box plot option is to be able to compare the metrics from different runs, usually equivalent runs. For example it makes sense to compare the metrics for the same dataset under different conditions but in the end is up to the user to decide what to compare. To that aim there is the option --target, to select a target DIR to compare with BASE_DIR. Kube will traverse the target DIR to find the dirs named with the timestamp and read the metrics. The options --since and --to are use to filter the search as you already know. There is an additional option --at to search for a specific date in the target DIR. You could achieve the same with --since and --to but using --at the syntax is more compact and readable.

When displaying a graph, if a single metric is shown, the absolute value is used. But if the user tries to display a combination of metrics into the same plot, Kube will use relative values to avoid scale representations artifacts. So, in this case the values form BASE_DIR are used as reference and the absolute value is shown in the axis.

Here are some examples:







Configuration file:

The configuration file is the core component of the framework. The whole process is driven by this file. It is very flexible and there are just a few rules to follow. This flexibility is an advantage and a downside at the same time since it brings you flexibility but the writer is responsible for the mistakes :) .

It has been written in YAML because of its simplicity, human readability and easy to parse. So anyone who open the config file could get a quick understanding of the process at a glance.

These are the sections in which the config file is divided:

Config file fields:

HOME:

path: Full path to the place where KUBE is installed

RUNS: An additional field

path: Path to the directory where the runs will be stored. If the leading "/" is provided then the absolute path is taken, otherwise the relative path to HOME will be used.

lifespan: Defines for how long the runs will be kept. This is an integer value representing days. If 0 keep runs results forever.

RESULTS:

path: Path to the directory where the results will be stored. If the leading "/" is provided then the absolute path is taken, otherwise the relative path to HOME will be used.

TOOLS:

path: Path to a special directory where the user can save scripts, binaries or anything that may be use to, for example, process the outputs. If the leading "/" is provided then the absolute path is taken, otherwise the relative path to HOME will be used.

RDDTOOL:

path: Path to where to store the databases

root: This is the path to where the rrdtool is installed. Where to find the binaries

BATCH: List of the available batch systems.

name: This is a mandatory field and is needed to identify the batch system to use within a benchmark.

script: The name of the template used to create the submission script. if the leading "/" is provided then the absolute path is taken, otherwise the relative path to [KUBE][HOME]/etc will be used. Example of the template script used for Noor. Notice that all *%<VAR_NAME>%* will be replaced by the specific dataset values at the moment of its creation:

```
#!/bin/bash -l

#BSUB -J %NAME%
#BSUB -o KUBE%NAME%_%NUMPROCS%.out
#BSUB -e KUBE_%NAME%_%NUMPROCS%.err
#BSUB -q %QUEUE%
#BSUB -W %WALLCLOCK%
#BSUB -n %NUMPROCS%
#BSUB %MPIFLAGS%
#BSUB -R span[ptile=%TASKS_PER_NODE%]

module purge
%MODULES%

%LAUNCHER% %LAUNCHER_FLAGS% %EXE% %ARGS%
```

submit:

submittedmsg: This is the output message (or just part of it) of the submission command containing the JOBID. The output of the submission command will be searched looking for this pattern to get the JOBID and also to determine if the job was submitted correctly to the batch system.

command: Submission command

parameters: Params to the submission commands if any

If the field 'script' is defined, then the submission command will be:

```
<command> <parameters> <script>
```

A special batch system named **MANUAL** is used in case there isn't any batch system available. This is a mandatory entry that will allow to execute jobs without any batch system. In this case the mandatory tags are **'name**' and **'submit**'. And inside submit, **'command**' and **'parameters**'. For a benchmark using manual launcher (**MANUAL** batch system) the submission command to be used would be:

```
<command> <parameters> <exe> <args>
```

Variables *%<VAR_NAME>%* specific to the dataset will be replaced at the moment the script is generated. Other *%<VAR_NAME>%* variables can be used in the **'submit**' section if they are defined... for example:

```
        - name: MANUAL
          hostslistfile: hostnames
          submit:
                command: mpirun
                parameters: -np %NUMPROCS% -machinefile %HOSTSLISTFILE%
```

In this example, **'hostslistfile**' has been added to be replaced in the **'parameters**' section. Notice that *%NUMPROCS%* will be replaced by the value of the field **'numproc**' of the current run.

The 'batch' field is mandatory for any dataset, but if the final user doesn't want to use any batch system or manual launcher, for example if you want to run just a sequential test like if you run it from the command line; then he just have to set **'batch**' to **'None**' in the desired dataset:

```
        - name: gpfs_iozone3
          active: true
          batch: None
          numprocs: 1
          modules:  module load gcc mpi-openmpi
          datasets:
               - name: iozone3_408
                 args: -Rab output.wks -g 2G  -I
                 exe: bin/iozone
                 dependencies: bin/*
                 active: true
                 outputs:
                   output: KUBE_%NAME%_%NUMPROCS%.out
                   iozone: iozone.tmp
```

BENCH: Contains the following fields: *APPS, FILESYSTEM, SYNTHETIC, NETWORKS*. Each of which contains a list of items to be benchmarked.

APPS: Contains a list with the applications to be used and its configuration. Mandatory fields for each apps are: **'name','batch','active','dataset**'

name: Identifies the app

batch: Selects the batch system to be used. Must be a valid name in the BATCH section.

active: Indicates whether the app will be run or not

dataset: Any active app must have a dataset defined. It is actually a list since one application could run several datasets. Outside the 'dataset' field, user can define whatever tag he wants and it will be used inside each dataset unless the same tag is defined inside any dataset. In this case the tag defined inside the dataset has preference. Note that this tags will be used to match the variables *%<VAR_NAME>%* (in capitals) in the submission script (or the submission command) and build up a usable script (or command). So there is no limitation on the tags, except the mandatory ones. The **'numprocs**' tag may contain a comma separated list of values and this tag represents the number of cores to be used

A special mention needs to be done to the **metrics** field since it is a very important component of Kube. This field is a list or array where element defines a metric to be computed for the dataset. Three mandatory fields are necessary and one optional. The mandatory fields are:

name: Defines the name of this metric

units: Gives more sense to the representation, just defines the units of what is being represented

command: Defines the command to be executed in order to compute the metric. It is important to point out here that the expected stream must have the form: <command> | <command> | ....

The optional field is *reference* which defines the range to validate the data computed.

reference:

value: It is a number or expression. It represents the actual reference value

tolerance: May accept the following values: *above* , *below* or *bilateral* and along with the next two fields, define a range of acceptance

threshold: Number or "Inf"

threshold_type: May accept *absolute* or *percentage*

Here is an example of an application section:

```
        - name: namd
          active: true
          batch: LSF
          # following parameters are dataset dependent...
          # This values will be used unless they are redefined in the "dataset" section
          exe: namd2_mpi
          modules:  module load gcc mpi-openmpi  libs-extra fftw2/2.1.5-openmpi-1.4.3-sse4.2-sp namd
          metrics:
             - name: wallclock
               units: secs
               command: cat %OUTPUT% | grep WallClock | gawk '{print $2}'
             - name: days/ns
               units: days/ns
               command: cat %OUTPUT% | grep  Benchmark | gawk 'END{print $8}'
               reference:
                      value: cat %OUTREF% | grep  Benchmark | gawk 'END{print $8}'
                      tolerance: above
                      threshold: inf
                      threshold_type:  absolute
             - name: cputime
               units: secs
               command: cat %OUTPUT% | grep CPUTime | gawk '{print $4}'
             - name: memory
               units: MB
               command: cat %OUTPUT%  | grep Memory | gawk '{print $6}'

          # here goes the specific dataset configuration
          datasets:
             - name: namd_nucleosome146Katoms
               active: false
               args: nucleosome146Katoms.namd  > nucleosome.out  2>nucleosome.err
               outputs:
                  output: nucleosome.out
                  another_output: KUBE_%NAME%_%NUMPROCS%.out

             - name: namd_apoa1
               active: true
               numprocs: 2,4
               args:  apoa1.namd   > apoa1.out 2>apoa1.err
               outputs:
                  output: apoa1.out
                  another_output: KUBE_%NAME%_%NUMPROCS%.out
```

The rest of tags inside **'BENCH**' are :

FILESYSTEMS:

NETWORKS:

SYNTHETICS:

They will be explained together since their behavior is similar.

The only differences of these fields compared with the **'APPS**' is that they may contain an additional tag: **'dependencies**' used to copy all the dependencies needed to create a self contained run directory. It allows wildcards and can be a comma separated list of items.

In addition to this, the element contained in **'exe'** will be copied to the execution site if not full path is specified.

For **'APPS**' it is understood that the dataset is self-contained, so no dependencies are allowed.

Some additional 'rules' to follow in the config file:

- Only **'outputs**' , **'args**', **'dependencies**' and **'exe**' fields may contain references to fields as *%<VAR_NAME>%*

- The basic and final element is a *dataset*. Any tag not defined at this level will be searched for in the immediate upper level: **'APPS**', **'FILESYSTEM**', ... and if not found there, then the corresponding **'BATCH**' entry is searched.

- if %NUMPROCS% is used, one entry per proc will be created.

- The **'metrics**' elements may contain references to the outputs names defined above as well as references to other tags in the config file.

- As a rule of thumb, all paths are relative unless the leading "/" is provided; then the absolute path is taken, otherwise the relative path is used.

- Any arbitrary field can be used

- The mandatory fields that must be defined when the dataset level is reached are: **name, active,args, exe, output** and **numprocs**