Advanced Apptainer Guide

Last updated 22 April, 2024 • 1 min read

This is a follow-up guide on Apptainer on how to customize containers for optimal productivity. For an introduction to Apptainer, please first read this article.

📘 Instructions

In this tutorial, you will learn

1. About `apptainer` definition files, which are informally termed recipes;
2. How to use pre-built docker images;
3. Use recipes to customize the pre-built docker images.

🧁 The basic of Apptainer recipes

The Apptainer recipe (or definition file) is a simple text file with a `.def` extension. These files are divided into two parts, the **Header** and the **Sections.**

1. The Header describes the operating system (OS) within the container.
2. The rest of the definition comprises Sections. Each section is defined by a `%` character followed by the name of the particular section. All sections are optional, and a `.def` file may contain more than one instance of a given section.

The Header

The header should be written at the top of your recipe, and it will need at least two "arguments," the `Bootstrap` agent and the `From`. Loosely speaking, the former tells Apptainer where to get the OS we desire to work with, while the latter will inform the specific OS and, optionally, its exact version. So, for example,

```
Bootstrap: docker
From: debian
```

will use the latest Debian OS as the basis for our container. The OS will be obtained from an official docker pre-built image. Alternatively, we can set up version 7 of Debian by replacing `From: debian` with `From: debian:7`.

The Sections

The Sections most commonly used are listed below.

%files

The `%files` section allows you to copy files into the container safely. Its general form lists the files (or directories) in the host and the container as follows:

```
%files
    /path/file1 /opt/

```

%post

In this section, you can download and install software from the internet as if in a command line of the chosen OS. For example, assuming we are running a Debian container, we can

- install `libomp-dev`, `wget`, and `git`;
- define a variable with the date and time when the container was built;
- clone a GitHub repository

as follows

```
 %post
    apt-get update && apt-get install -y \
      libomp-dev \
      git
    export NOW=`date`
    git clone https://github.com/tatsu-lab/stanford_alpaca.gi

```

%environment

The `%environment` section allows you to define environment variables available at runtime. If you define a variable in `%post` it will be available when building the container but not when executing it. Conversely, variables included in the `%environment` section are unavailable at build time. For instance, to make the previously created `NOW` variable available at runtime as well, we should define the `%environment` section as follows:

```
%environment
      NOW=$NOW
```

%runscript

The contents of the `%runscript` section are executed when the container image is run using, for example, `apptainer run`. The following `%runscript` section displays the time when the container was created.

```
%runscript
    echo "Container was created $NOW"
```

%labels

This section creates metadata for your container, and its general format is a name-value pair. Below, we are defining the author and the version of our container.

```
%labels
    Author hpc@uconn
    Version v0.1
```

%help

The `%help` section helps describe the purpose of the container and how a user can interact with it. Its content will be incorporated into the container metadata with the content from `%labels`. The `%help` content can be retrieved using the `run-help` command.

Below we provide a simple help section:

```
  %help
      This container purpose is to describe common sections f
```

You can find a complete list of available Bootstrap agents and more details about the Sections in Apptainer's official documentation.

In the following section of this tutorial, we put into practice the knowledge we just acquired about the sections of Apptainer recipes.

🐳 Using a pre-built image from Docker to build a

custom `geopandas` container

Let us call `geopandas.def` the definition file containing the chunk of code below. The provided commands will install the `geopandas` python library on the top of a GDAL pre-built image. This is convenient because GDAL is a “tricky to install” dependency for the `geopandas` library. Our `%runscript` will show the time at which the container was built and, in addition, the operating system used by the container.

```
Bootstrap: docker
From: osgeo/gdal:ubuntu-full-3.6.3

%help
    This container is intended to use a pre-built gdal image
   `geopandas` python library.

%post
    curl -sSL https://bootstrap.pypa.io/get-pip.py -o get-pip
    python3 get-pip.py && rm get-pip.py
    pip install geopandas

%runscript
    echo "The geopandas version is `pip freeze | grep geopanc

%labels
    Author hpc@uconn
    Version v0.1

```

To build the container, we will load the `apptainer` module and then use the `apptainer build` command as follows:

```
module load apptainer
apptainer build geopandas.sif geopandas.def

```

To see the metadata associated with a container run the following:

```
apptainer inspect geopandas.sif

```

We can use `apptainer run` as follows to execute our `runscript`

```
apptainer run geopandas.sif

```

✍️ Summary

- Choose a pre-built image (usually from the Docker hub).
- Use `%files` to transfer local files to the container (if needed).
- In the `%post` section, you can use `bash` to install the software you need in your container.

📋 Example definition file used to setup Ansys Rocky through Apptainer and apptainer command to call Ansys Rocky

```
Bootstrap: docker
From: centos:7

%post

yum install -y squashfs-tools
yum install -y fontconfig
yum install -y mlocate
yum install -y rsync
yum install -y pciutils
yum install -y alsa-lib
#yum install -y nss-tools
#yum install -y firefox
yum install -y cmake pkg-config
yum install -y xorg-x11-server-Xorg xorg-x11-xauth xorg-x11-a
yum install -y mesa-libGL-devel mesa-libGLU mesa-dri-drivers
yum install -y qt5-qtbase qt5-qtbase-devel
yum install -y glx-utils

#export NOW=`date`
#export TZ=America/New_York
#export LANGUAGE=en_US.UTF-8
#export LC_ALL=en_US.UTF-8
#export LANG=en_US.UTF-8
#export LC_CTYPE=en_US.UTF-8
#export LSTC_LICENSE=network
#export LSTC_LICENSE_SERVER=1055@engr-license3.engr.uconn.edu
#export ANSYS241_DIR=/gpfs/sharedfs1/admin/hpc2.0/apps/ansys/
#export ANSYSLMD_LICENSE_FILE=/gpfs/sharedfs1/admin/hpc2.0/ap
#export QT_DEBUG_PLUGINS=1

%environment
export NOW=$NOW
export LANGUAGE=en_US.UTF-8
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
export LC_CTYPE=en_US.UTF-8

export LSTC_LICENSE=network
export LSTC_LICENSE_SERVER=1055@engr-license3.engr.uconn.edu
#export ANSYS241_DIR=/gpfs/sharedfs1/admin/hpc2.0/apps/ansys/
export ANSYSLMD_LICENSE_FILE=/gpfs/sharedfs1/admin/hpc2.0/app
export ANSYS241_DIR=/ansys/v241
export AWP_ROOT241=/ansys/v241
```

```

%runscript
echo "Container was created $NOW"
echo $LANGUAGE
echo $LC_ALL
echo $LANG
echo $LC_CTYPE
echo $LSTC_LICENSE
echo $LSTC_LICENSE_SERVER
echo $ANSYS241_DIR
echo $ANSYSLMD_LICENSE_FILE
echo $AWP_ROOT241

%labels
Author hpc@uconn
Version v0.1

```

An interactive GPU job needs to be submitted, the apptainer module loaded, and the following apptainer command to call Ansys Rocky:

```
apptainer exec --nv --unsquash -H $HOME:/home -B /gpfs/sharec
```

Errors will show up, but say Y to the prompt Ansys Rocky gives about overwriting:

```
mv: try to overwrite ‘/ansys/v241/rocky/Rocky.desktop’, overr
```

If other errors show up, ignore the errors and rerun the above apptainer command a couple of more times until Ansys Rocky loads. The errors are Red Herring errors and can be ignored.

📋 Related articles

Apptainer Guide
18 Apr, 2025

Advanced Apptainer Guide
22 Apr, 2024