Bright Cluster Manager 9.0

Machine Learning Manual

Revision: f978b5e

Date: Sat Sep 26 2020

Trademarks

Linux is a registered trademark of Linus Torvalds. PathScale is a registered trademark of Cray, Inc. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. SUSE is a registered trademark of Novell, Inc. PGI is a registered trademark of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. PBS Professional, PBS Pro, and Green Provisioning are trademarks of Altair Engineering, Inc. All other trademarks are the property of their respective owners.

Rights and Restrictions

All statements, specifications, recommendations, and technical information contained herein are current or planned as of the date of publication of this document. They are reliable as of the time of this writing and are presented without warranty of any kind, expressed or implied. Bright Computing, Inc. shall not be liable for technical or editorial errors or omissions which may occur in this document. Bright Computing, Inc. shall not be liable for any damages resulting from the use of this document.

Limitation of Liability and Damages Pertaining to Bright Computing, Inc.

The Bright Cluster Manager product principally consists of free software that is licensed by the Linux authors free of charge. Bright Computing, Inc. shall have no liability nor will Bright Computing, Inc. provide any warranty for the Bright Cluster Manager to the extent that is permitted by law. Unless confirmed in writing, the Linux authors and/or third parties provide the program as is without any warranty, either expressed or implied, including, but not limited to, marketability or suitability for a specific purpose. The user of the Bright Cluster Manager product shall accept the full risk for the quality or performance of the product. Should the product malfunction, the costs for repair, service, or correction will be borne by the user of the Bright Cluster Manager product. No copyright owner or third party who has modified or distributed the program as permitted in this license shall be held liable for damages, including general or specific damages, damages caused by side effects or consequential damages, resulting from the use of the program or the un-usability of the program (including, but not limited to, loss of data, incorrect processing of data, losses that must be borne by you or others, or the inability of the program to work together with any other program), even if a copyright owner or third party had been advised about the possibility of such damages unless such copyright owner or third party has signed a writing to the contrary.

Table of Contents

Preface

Welcome to the *Machine Learning Manual* for Bright Cluster Manager 9.0.

0.1 About This Manual

This manual is aimed at helping cluster administrators install, understand, configure, and manage basic machine learning capabilities easily using Bright Cluster Manager. The administrator is expected to be reasonably familiar with the *Administrator Manual*.

0.2 About The Manuals In General

Regularly updated versions of the Bright Cluster Manager 9.0 manuals are available on updated clusters by default at `/cm/shared/docs/cm`. The latest updates are always online at `http://support.brightcomputing.com/manuals`.

- The *Installation Manual* describes installation procedures for a basic cluster.
- The *Administrator Manual* describes the general management of the cluster.
- The *User Manual* describes the user environment and how to submit jobs for the end user.
- The *Cloudbursting Manual* describes how to deploy the cloud capabilities of the cluster.
- The *Developer Manual* has useful information for developers who would like to program with Bright Cluster Manager.
- The *OpenStack Deployment Manual* describes how to deploy OpenStack with Bright Cluster Manager.
- The *Edge Manual* describes how to deploy Bright Edge with Bright Cluster Manager.
- The *Machine Learning Manual*—this manual—describes how to install and configure machine learning capabilities with Bright Cluster Manager.

If the manuals are downloaded and kept in one local directory, then in most pdf viewers, clicking on a cross-reference in one manual that refers to a section in another manual opens and displays that section in the second manual. Navigating back and forth between documents is usually possible with keystrokes or mouse clicks.

For example: `<Alt>-<Backarrow>` in Acrobat Reader, or clicking on the bottom leftmost navigation button of xpdf, both navigate back to the previous document.

The manuals constantly evolve to keep up with the development of the Bright Cluster Manager environment and the addition of new hardware and/or applications. The manuals also regularly incorporate customer feedback. Administrator and user input is greatly valued at Bright Computing. So any comments, suggestions or corrections will be very gratefully accepted at `manuals@brightcomputing.com`.

There is also a feedback form available via Bright View, via the Account icon,  , following the clickpath:

`Account`→`Help`→`Feedback`

0.3 Getting Administrator-Level Support

If the reseller from whom Bright Cluster Manager was bought offers direct support, then the reseller should be contacted.

Otherwise the primary means of support is via the website `https://support.brightcomputing.com`. This allows the administrator to submit a support request via a web form, and opens up a trouble ticket. It is a good idea to try to use a clear subject header, since that is used as part of a reference tag as the ticket progresses. Also helpful is a good description of the issue. The followup communication for this ticket goes via standard e-mail. Section 13.2 of the *Administrator Manual* has more details on working with support.

0.4 Getting Professional Services

Bright Computing normally differentiates between professional services (customer asks Bright Computing to do something or asks Bright Computing to provide some service) and support (customer has a question or problem that requires an answer or resolution). Professional services can be provided after consulting with the reseller, or the Bright account manager.

1

Introduction And Machine Learning Packages Installation

1.1 Introduction

From Bright Cluster Manager version 7.3 onward, a number of machine learning and deep learning library and framework packages can be used. The packages provided make it faster and easier for organizations to install the latest state-of-the-art libraries, and gain insights from rich, complex data.

1.1.1 Bright Cluster Manager Versions—Associated Repositories And Support For Machine Learning Packages

- In Bright Cluster Manager versions 7.3 and 8.0 the machine learning and deep learning packages are experimentally accessible to a cluster from the standard Bright `cm` repository. However, from Bright Cluster Manager version 8.1 onward, these packages are distributed via a separate Bright `cm-ml` repository.
- In Bright Cluster Manager versions 8.1 and 8.2, cluster administrators have to activate the Data Science Add-on using the online wizard at:

  `http://licensing.brightcomputing.com/licensing/activate-data-science/`

  The add-on enables the dedicated Bright `cm-ml` repository.
- From Bright Cluster Manager version 9.0 onward the activation step via a wizard is not needed—the repository is automatically available and enabled.

Only if the cluster is licensed for the Data Science Add-on package does Bright provide support for the integration of packages distributed in the dedicated repository with Bright Cluster Manager.

For convenience, a summary of the Bright for Data Science (B4DS) repository configuration requirements is shown in table 1.1:

Table 1.1: Bright Cluster Manager offering overview for Machine Learning packages

Bright CM	Packages repository	Repository configuration	Support
7.3	Traditional (cm)	Automatically available and enabled	Not available
8.0	Traditional (cm)	Automatically available and enabled	Not available
8.1	Dedicated (cm-ml)	Requires B4DS Add-on activation*	Requires B4DS Add-on
8.2	Dedicated (cm-ml)	Requires B4DS Add-on activation*	Requires B4DS Add-on
9.0	Dedicated (cm-ml)	Automatically available and enabled	Requires B4DS Add-on

* Activation is via http://licensing.brightcomputing.com/licensing/activate-data-science/

An administrator who is upgrading a cluster that has machine learning and deep learning packages installed on it should always make sure that the dedicated Bright cm-ml repository is accessible, if required by the new Bright Cluster Manager version.

1.1.2 Bright Cluster Manager Versions—Supported Distributions And Architecture For Machine Learning Packages

At the time of writing, February 2020, a number of different Linux distribution versions and architectures are supported, depending on the Bright Cluster Manager version. For convenience, a support matrix for this is shown in table 1.2:

Table 1.2: Supported Linux distributions and architectures for Bright Machine Learning packages

Bright CM	Architectures	Linux distributions
7.3	x86_64	CentOS 7, RHEL 7, SLES 12
8.0	x86_64	CentOS 7, RHEL 7, SLES 12
8.1	x86_64	CentOS 7, RHEL 7, SLES 12
8.2	x86_64	CentOS 7, RHEL 7, SLES 12, Ubuntu 18.04
9.0	x86_64	CentOS 7, CentOS 8, RHEL 7, RHEL 8, SLES 12, SLES 15, Ubuntu 18.04

An updated list of the supported Linux distributions and architectures available for the various Bright Cluster Manager versions can be found at https://support.brightcomputing.com/feature-matrix/, in the section of the Feature column dedicated to machine learning.

1.2 Available Packages

An updated list of the machine learning packages available for the various Bright Cluster Manager versions can be found at https://support.brightcomputing.com/packages-dashboard/. Most of the machine learning packages are to be found within the ml group. However, some of them are found within the cm group for legacy reasons.

At the time of writing, February 2020, the following packages were available:

Table 1.3: Machine Learning packages provided by the Bright Cluster Manager repositories

Package name	Description
cm-bazel	An open-source build and test tool similar to Make, Maven, and Gradle. It uses a human-readable, high-level build language. Bazel supports projects in multiple languages and builds outputs for multiple platforms.
cm-chainer-py36-cuda10.1-gcc	A flexible framework for neural networks, designed to write complex architectures simply and intuitively.
cm-chainer-py37-cuda10.1-gcc	A flexible framework for neural networks, designed to write complex architectures simply and intuitively.
cm-cub-cuda10.1	A flexible library of cooperative threadblock primitives and other utilities for CUDA kernel programming.
cm-cub-cuda10.2	A flexible library of cooperative threadblock primitives and other utilities for CUDA kernel programming.
cm-dynet-py36-cuda10.1-gcc	A neural network library developed by Carnegie Mellon University and many others. It is written in C++ (with bindings in Python) and is designed to be efficient when run on either CPU or GPU, and to work well with networks that have dynamic structures that change for every training instance.
cm-dynet-py37-cuda10.1-gcc	A neural network library developed by Carnegie Mellon University and many others. It is written in C++ (with bindings in Python) and is designed to be efficient when run on either CPU or GPU, and to work well with networks that have dynamic structures that change for every training instance.
cm-fastai-py36-cuda10.1-gcc	A library that simplifies training fast and accurate neural nets using modern best practices.
cm-fastai-py37-cuda10.1-gcc	A library that simplifies training fast and accurate neural nets using modern best practices.
cm-gpytorch-py36-cuda10.1-gcc	A Gaussian process library implemented using PyTorch.
cm-gpytorch-py37-cuda10.1-gcc	A Gaussian process library implemented using PyTorch.
cm-horovod-mxnet-py36-cuda10.1-gcc	A distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.

...continues

Table 1.3: Machine Learning Packages Included...continued

Package name	Description
cm-horovod-mxnet-py37-cuda10.1-gcc	A distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.
cm-horovod-pytorch-py36-cuda10.1-gcc	A distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.
cm-horovod-pytorch-py37-cuda10.1-gcc	A distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.
cm-horovod-tensorflow-py36-cuda10.1-gcc	A distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.
cm-horovod-tensorflow-py37-cuda10.1-gcc	A distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.
cm-keras-py36-cuda10.1-gcc	A high-level neural networks Python API, capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.
cm-keras-py36-mkl-gcc8	A high-level neural networks Python API, capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.
cm-keras-py37-cuda10.1-gcc	A high-level neural networks Python API, capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.
cm-keras-py37-cuda10.2-gcc	A high-level neural networks Python API, capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.
cm-keras-py37-mkl-gcc8	A high-level neural networks Python API, capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.
cm-ml-distdeps-cuda10.1	Meta-package containing distribution-specific dependencies for machine learning frameworks.
cm-ml-distdeps-cuda10.2	Meta-package containing distribution-specific dependencies for machine learning frameworks.
cm-ml-distdeps-mkl	Meta-package containing distribution-specific dependencies for machine learning frameworks.
cm-ml-pythondeps-py36-cuda10.1-gcc	Meta-package containing Python dependencies for machine learning frameworks. It includes, for example, NumPy, PyCUDA, pandas, NetworkX, pydot and Matplotlib.

...continues

Table 1.3: Machine Learning Packages Included...continued

Package name	Description
cm-ml-pythondeps-py36-mkl-gcc8	Meta-package containing Python dependencies for machine learning frameworks. It includes, for example, NumPy, PyCUDA, pandas, NetworkX, pydot and Matplotlib.
cm-ml-pythondeps-py37-cuda10.1-gcc	Meta-package containing Python dependencies for machine learning frameworks. It includes, for example, NumPy, PyCUDA, pandas, NetworkX, pydot and Matplotlib.
cm-ml-pythondeps-py37-cuda10.2-gcc	Meta-package containing Python dependencies for machine learning frameworks. It includes, for example, NumPy, PyCUDA, pandas, NetworkX, pydot and Matplotlib.
cm-ml-pythondeps-py37-mkl-gcc8	Meta-package containing Python dependencies for machine learning frameworks. It includes, for example, NumPy, PyCUDA, pandas, NetworkX, pydot and Matplotlib.
cm-mxnet-py36-cuda10.1-gcc	A deep learning framework designed for both efficiency and flexibility. It allows a mix of symbolic and imperative programming and contains a dynamic dependency scheduler that automatically parallelizes any operations on the fly.
cm-mxnet-py37-cuda10.1-gcc	A deep learning framework designed for both efficiency and flexibility. It allows a mix of symbolic and imperative programming and contains a dynamic dependency scheduler that automatically parallelizes any operations on the fly.
cm-nccl2-cuda10.1-gcc	Version 2 of NCCL, an implementation of multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs.
cm-nccl2-cuda10.2-gcc	Version 2 of NCCL, an implementation of multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs.
cm-opencv3-py36-cuda10.1-gcc	An open-source BSD-licensed library that includes several hundreds of computer vision algorithms.
cm-opencv3-py36-mkl-gcc8	An open-source BSD-licensed library that includes several hundreds of computer vision algorithms.
cm-opencv3-py37-cuda10.1-gcc	An open-source BSD-licensed library that includes several hundreds of computer vision algorithms.
cm-opencv3-py37-cuda10.2-gcc	An open-source BSD-licensed library that includes several hundreds of computer vision algorithms.
cm-opencv3-py37-mkl-gcc8	An open-source BSD-licensed library that includes several hundreds of computer vision algorithms.

...continues

Table 1.3: Machine Learning Packages Included...continued

Package name	Description
cm-protobuf3-gcc	Version 3 of Protocol Buffers, a language-neutral, platform-neutral extensible mechanism for serializing structured data.
cm-protobuf3-gcc8	Version 3 of Protocol Buffers, a language-neutral, platform-neutral extensible mechanism for serializing structured data.
cm-pytorch-py36-cuda10.1-gcc	An optimized tensor library for deep learning using GPUs and CPUs. It provides tensor computation (like NumPy) with strong GPU acceleration, and deep neural networks built on a tape-based autograd system. It now includes Caffe2.
cm-pytorch-py37-cuda10.1-gcc	An optimized tensor library for deep learning using GPUs and CPUs. It provides tensor computation (like NumPy) with strong GPU acceleration, and deep neural networks built on a tape-based autograd system. It now includes Caffe2.
cm-pytorch-py37-cuda10.2-gcc	An optimized tensor library for deep learning using GPUs and CPUs. It provides tensor computation (like NumPy) with strong GPU acceleration, and deep neural networks built on a tape-based autograd system. It now includes Caffe2.
cm-tensorflow-py36-cuda10.1-gcc	An open source software library for numerical computation using data flow graphs originally developed by researchers and engineers working on the Google Brain team.
cm-tensorflow-py36-mkl-gcc8	An open source software library for numerical computation using data flow graphs originally developed by researchers and engineers working on the Google Brain team.
cm-tensorflow-py37-cuda10.1-gcc	An open source software library for numerical computation using data flow graphs originally developed by researchers and engineers working on the Google Brain team.
cm-tensorflow-py37-mkl-gcc8	An open source software library for numerical computation using data flow graphs originally developed by researchers and engineers working on the Google Brain team.
cm-tensorflow2-py36-cuda10.1-gcc	An open source software library for numerical computation using data flow graphs originally developed by researchers and engineers working on the Google Brain team.
cm-tensorflow2-py37-cuda10.1-gcc	An open source software library for numerical computation using data flow graphs originally developed by researchers and engineers working on the Google Brain team.
cm-tensorflow2-py37-cuda10.2-gcc	An open source software library for numerical computation using data flow graphs originally developed by researchers and engineers working on the Google Brain team.

...continues

Table 1.3: Machine Learning Packages Included...continued

Package name	Description
cm-tensorrt-cuda10.1-gcc	A platform for high-performance deep learning inference designed by NVIDIA and built on CUDA. It includes a deep learning inference optimizer and runtime that delivers low latency and high-throughput for deep learning inference applications.
cm-tensorrt-cuda10.2-gcc	A platform for high-performance deep learning inference designed by NVIDIA and built on CUDA. It includes a deep learning inference optimizer and runtime that delivers low latency and high-throughput for deep learning inference applications.
cm-theano-py36-cuda10.1-gcc	A Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.
cm-theano-py37-cuda10.1-gcc	A Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.
cm-xgboost-py36-cuda10.1-gcc	An optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the gradient boosting framework.
cm-xgboost-py37-cuda10.1-gcc	An optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the gradient boosting framework.

Legend:

Package are available for every distribution unless otherwise tagged.

1.2.1 Considerations

There are some considerations that the cluster administrator should be aware of with the packages.

- Some packages may be labelled in the table 1.3 as deprecated. "Deprecated" in the software industry is not a well-defined term. Here it is used by Bright Computing to mean a package that may no longer be offered in a future release, or for which a newer existing version is preferred.
- Several different packages may be provided for the same machine learning library or framework. For example, TensorFlow may be provided by:
  - `cm-tensorflow-py36-cuda10.1-gcc` or
  - `cm-tensorflow-py37-mkl-gcc8`

  As is the norm with other package management systems in the software industry, the name given to a Bright Computing package includes the most relevant dependencies required to build and use it. The dependencies commonly highlighted in this manner are:
  - Python interpreter version used (e.g. `*-py36*` and `*-py37*`)
  - accelerator library used (e.g. `*-cuda10.1*` and `*-mkl*`)
  - compiler used (e.g. `*-gcc*` and `*-gcc8*`)

  The availability of different variants of the same package makes it easier for administrators to set up a working environment that is suited to their needs.
- Machine learning packages are designed to coexist, and can therefore all be installed at the same time. This also applies to different variants of the same library or framework.

This means that administrators can install different versions of the same machine learning library or framework, simply by using different variants. For example: an older `*-py36*` version of TensorFlow, as well as a more recent `*-py37*` version of TensorFlow, can both be installed at the same time.

- As is done with other packages provided by Bright Computing, the updates released for machine learning libraries and frameworks generally leave their major versions unchanged.

  Whenever a major version for a third party machine learning library or a framework is publicly released, a new package or a set of packages is typically placed in the repository.

  Such package(s) imply or contain a reference to the major version in the name. For example:

  - `cm-tensorflow-*` is the name used for TensorFlow major version 1
  - `cm-tensorflow2-*` is the name for TensorFlow major version 2

  As a result, administrators can safely upgrade cluster packages without breaking backward compatibility with users' applications.

- MPI modules and libraries should not be blindly added by the cluster administrator. During module loading, warnings are typically given to suggest an MPI library (Open MPI, or an MPICH or MVAPICH implementation of Open MPI) is required. However, the exact implementation of the MPI library that can be used depends upon the hardware (GPU, interface, architecture) used and requires judgment of suitability by the cluster administrator. Bright Cluster Manager uses the `cm-openmpi-geib-cuda10.1-gcc` package in this manual as the reference MPI library implementation. This driver package corresponds with using Open MPI with Gigabit Ethernet and InfiniBand networking, NVIDIA GPUs, and x86 architecture.

1.3 Requirements

The following requirements must be met before installing the preceding machine learning packages.

- RHEL users must have access to the YUM repositories and EPEL repository.
- There must be enough free space for the packages that are installed on the head node and compute nodes. The actual amount depends on the packages installed.
- 8 GB of RAM on the nodes is the minimum recommended amount.
- In order to use packages built with the CUDA toolkit accelerator library (i.e. `*-cuda*`), the NVIDIA GPUs must be Maxwell or more recent, with compute capability 3.5 or later. CUDA compute capability 6.0 or later is recommended.
- In order to use packages built with GCC 5 as compiler (i.e. `*-gcc`), the CPU must support the AVX/AVX2, FMA, and SSE4.2 instructions. This can be checked by inspecting the CPU flags:

  Example

  ```
  [root@node ~]# egrep -m1 -o '(avx|avx2|fma|sse4_2)' /proc/cpuinfo
  fma
  sse4_2
  avx
  avx2
  ```

- In order to use packages built using GCC 8 as the compiler (i.e. `*-gcc8`), the CPU must support the AVX-512 Vector Neural Network Instructions (VNNI). Examples of such CPUs are Intel Xeon Scalable processors with Deep Learning Boost.

1.4 Machine Learning Packages Installation

Head Node Installation

Bright Cluster Manager machine learning packages are installed in the `/cm/shared` directory, which is by default exported over NFS. Packages installed on the head node are therefore also available to all the compute nodes by default.

The `.rpm` and `.deb` files have proper dependencies defined. This means that the cluster administrator does not need to spend time figuring out what needs to be installed to set up a working environment. Whenever a package is installed or updated, the required dependencies will be also automatically fetched, if necessary. As a result, packages can be installed with the usual package manager that is provided by the Linux distribution in the usual way (page 433 of the *Administrator Manual*).

For example, the administrator can install `cm-pytorch-py36-cuda10.1-gcc` as follows:

Example

```
[root@bright90 ~]# yum install cm-pytorch-py36-cuda10.1-gcc       #on RHEL 7

[root@bright90 ~]# zypper install cm-pytorch-py36-cuda10.1-gcc    #on SLES 15

[root@bright90 ~]# apt-get install cm-pytorch-py36-cuda10.1-gcc   #on Ubuntu 18.04
```

The package managers also automatically install the corresponding dependencies, such as

- `cm-ml-distdeps-cuda10.1`
- `cm-ml-pythondeps-py36-cuda10.1-gcc`
- `cm-protobuf3-gcc`
- `cuda10.1-toolkit`
- `cm-cudnn7.6-cuda10.1`
- `cm-nccl2-cuda10.1-gcc`

Machine learning packages share several dependencies, usually providing useful Python or system libraries. For convenience, these dependencies are grouped in the `cm-ml-pythondeps-*` and `cm-ml-distdeps-*` meta-packages.

- `cm-ml-pythondeps-*`: This meta-package provides the application libraries such as `numba`, `numpy`, `scikit-learn`, and `scipy`.
- `cm-ml-distdeps-*`: This meta-package, on the other hand, provides development libraries such as `blas-devel`, `libjpeg-devel` and `libpng-devel`, and the utility library `gnuplot`.

The appropriate meta-packages are automatically installed whenever a package installation requires it.

Administrators only need to make sure that their clusters meet the preceding hardware requirements listed at the start of section 1.3. If that is not done, then unexpected failures may occur during run time, such as segmentation faults.

Examples of common mistakes are

- installing packages requiring CUDA (e.g. `cm-pytorch-py36-cuda10.1-gcc`) on clusters without GPUs
- installing packages requiring VNNI (e.g. `cm-tensorflow-py36-mkl-gcc8`) on CPUs not supporting the instruction set

Compute Nodes Installation

The `cm-ml-distdeps-*` meta-packages must be also installed onto all compute nodes that are to run machine learning applications.

For example, if the name of the software image is `gpu-image`, then the administrator can install `cm-ml-distdeps-cuda10.1` on RHEL 7 as follows:

Example

```
[root@bright90 ~]# yum install --installroot=/cm/images/gpu-image cm-ml-distdeps-cuda10.1
```

The preceding command must be applied to all software images that are used to run machine learning applications.

There are equivalents to the `--installroot` option of `yum` for the other distribution package managers.

For SLES the installation command equivalent is:

```
[root@bright90 ~]# zypper --root /cm/images/gpu-image install cm-ml-distdeps-cuda10.1
```

For Ubuntu the installation command equivalent is:

```
[root@bright90 ~]# cm-chroot-sw-img /cm/images/gpu-image
[root@bright90 ~]# apt install cm-ml-distdeps-cuda10.1
[root@bright90 ~]# exit     #get out of chroot
```

Details on using `zypper` and `apt` commands for installation to software images are given on page 11.2.1 of the *Administrator Manual*.

The preceding command must be applied to all software images that are used to run machine learning applications.

1.4.1 Module Loading

Bright Cluster Manager provides environment module definitions for all the machine learning packages. The environment module files are also compatible with the Lmod software introduced in Bright Cluster Manager 7.3. They can be listed once the `shared` module is loaded, if it has not already been loaded:

```
[root@bright90 ~]# module purge; module available
---------------------------------- /cm/local/modulefiles ----------------------------------
cluster-tools/9.0         cmd    freeipmi/1.6.4   luajit       openldap
cm-image/9.0              cmjob  gcc/9.2.0        module-git   python3
cm-scale/cm-scale.module  cmsh   ipmitool/1.8.18  module-info  python37
cm-setup/9.0              dot    lua/5.3.5        null         shared
[root@bright90 ~]# module load shared; module available
---------------------------------- /cm/local/modulefiles ----------------------------------
cluster-tools/9.0         cmd    freeipmi/1.6.4   luajit       openldap
cm-image/9.0              cmjob  gcc/9.2.0        module-git   python3
cm-scale/cm-scale.module  cmsh   ipmitool/1.8.18  module-info  python37
cm-setup/9.0              dot    lua/5.3.5        null         shared

---------------------------------- /cm/shared/modulefiles ----------------------------------
blacs/openmpi/gcc/64/1.1patch03    hpl/2.2
blas/gcc/64/3.8.0                  hwloc/1.11.11
bonnie++/1.98                      intel-tbb-oss/ia32/2019_20191006oss
cm-image/master                    intel-tbb-oss/intel64/2019_20191006oss
cm-pmix3/3.1.4                     iozone/3_487
cuda10.1/blas/10.1.243             keras-py36-cuda10.1-gcc/2.3.1
cuda10.1/fft/10.1.243              lapack/gcc/64/3.8.0
cuda10.1/nsight/10.1.243           ml-pythondeps-py36-cuda10.1-gcc/3.0.0
```

```
cuda10.1/profiler/10.1.243          mpich/ge/gcc/64/3.3
cuda10.1/toolkit/10.1.243           mvapich2/gcc/64/2.3
cudnn7.6-cuda10.1/7.6.5.32          nccl2-cuda10.1-gcc/2.4.8
default-environment                 netcdf/gcc/64/4.6.1
fftw2/openmpi/gcc/64/double/2.1.5   netperf/2.7.0
fftw2/openmpi/gcc/64/float/2.1.5    openblas/dynamic/(default)
fftw3/openmpi/gcc/64/3.3.8          openblas/dynamic/0.2.20
gcc5/5.5.0                          openmpi/gcc/64/1.10.7
gdb/8.3.1                           protobuf3-gcc/3.10.1
globalarrays/openmpi/gcc/64/5.7     scalapack/openmpi/gcc/64/2.0.2
hdf5/1.10.1                         tensorflow-py36-cuda10.1-gcc/1.14.0
hdf5_18/1.8.20                      ucx/1.6.1
```

For example, after having installed the `cm-tensorflow-py36-cuda10.1-gcc` package, the associated TensorFlow module can be loaded with:

```
[root@bright90 ~]# module load tensorflow-py36-cuda10.1-gcc
Loading tensorflow-py36-cuda10.1-gcc/1.14.0
  Loading requirement: gcc5/5.5.0 python36 cuda10.1/toolkit/10.1.243
    ml-pythondeps-py36-cuda10.1-gcc/3.0.0 openblas/dynamic/0.2.20 cudnn7.6-cuda10.1/7.6.5.32
    hdf5_18/1.8.20 keras-py36-cuda10.1-gcc/2.3.1 nccl2-cuda10.1-gcc/2.4.8
```

The machine learning environment modules automatically load additional environment modules as dependencies, with the notable exception of Open MPI implementations for the reasons given in section 1.2.1.

The module dependencies are achieved via the module definition files:

Example

```
[root@bright90 ~]# module show tensorflow-py36-cuda10.1-gcc
-------------------------------------------------------------------
/cm/shared/modulefiles/tensorflow-py36-cuda10.1-gcc/1.15.2:

module-whatis    adds TensorFlow to your environment variables
module           load ml-pythondeps-py36-cuda10.1-gcc
module           load keras-py36-cuda10.1-gcc
module           load cuda10.1/toolkit
module           load openblas
module           load protobuf3-gcc
module           load cudnn7.6-cuda10.1
module           load hdf5_18
module           load nccl2-cuda10.1-gcc
module           load gcc5
prepend-path     PYTHONPATH /cm/shared/apps/tensorflow-py36-cuda10.1-gcc/1.15.2/lib/python3.6/site-packages/
prepend-path     PYTHONPATH /cm/shared/apps/tensorflow-py36-cuda10.1-gcc/1.15.2/lib64/python3.6/site-packages/
prepend-path     LD_LIBRARY_PATH /cm/shared/apps/tensorflow-py36-cuda10.1-gcc/1.15.2/lib/python3.6/\
site-packages/tensorflow_core
prepend-path     LD_LIBRARY_PATH /cm/shared/apps/tensorflow-py36-cuda10.1-gcc/1.15.2/lib/python3.6/\
site-packages/tensorflow_core/include
prepend-path     LD_LIBRARY_PATH /cm/shared/apps/tensorflow-py36-cuda10.1-gcc/1.15.2/lib64/python3.6/\
site-packages/tensorflow_core
prepend-path     LD_LIBRARY_PATH /cm/shared/apps/tensorflow-py36-cuda10.1-gcc/1.15.2/lib64/python3.6/\
site-packages/tensorflow_core/include
prepend-path     PATH /cm/shared/apps/tensorflow-py36-cuda10.1-gcc/1.15.2/bin
-------------------------------------------------------------------
```

1.5 Machine Learning Packages Removal

Machine learning packages can be removed in the usual way with the package manager commands used by the Linux distribution. For example, the administrator can remove the `cm-pytorch-py36-cuda10.1-gcc` package with:

Example

```
[root@bright90 ~]# yum remove cm-pytorch-py36-cuda10.1-gcc       #on RHEL 7
```

or

```
[root@bright90 ~]# zypper remove cm-pytorch-py36-cuda10.1-gcc    #on SLES 15
```

or

```
[root@bright90 ~]# apt-get remove cm-pytorch-py36-cuda10.1-gcc   #on Ubuntu 18.04
```

Bright Cluster Manager machine learning packages are installed in the `/cm/shared` directory, which is by default exported over NFS. Packages removed from the head node are therefore also removed from all the compute nodes by default.

The `cm-ml-distdeps-*` meta-packages must be also removed from all compute nodes that are to run machine learning applications. For example, if the name of the software image is `gpu-image`, then the images directory is `/cm/images/gpu-image`, and then the administrator can remove `cm-ml-distdeps-cuda10.1` from the image as follows in RHEL 7:

Example

```
[root@bright90 ~]# yum remove --installroot=/cm/images/gpu-image cm-ml-distdeps-cuda10.1
```

The preceding command must be applied to all software images that are used to run the machine learning applications.

The equivalents to the `--installroot` option of `yum` for the other distribution package managers are described in section 1.4.

2

Running TensorFlow

This chapter goes through some example runs with TensorFlow. Some output messages have been removed in the runs for readability.

It assumes TensorFlow has been installed, with, for example, `yum install cm-tensorflow-py36-cuda10.1-gcc`.

In addition, TensorFlow requires an OpenMPI implementation to work. As described in Chapter 3 of the *User Manual*, the Bright repositories provide different OpenMPI packages. This allows the user to choose which one to use, depending on, for example, which interconnect is being used, or if CUDA support is required.

In this chapter, the `cm-openmpi-geib-cuda10.1-gcc` package is used.

2.1 Hello World

A "`Hello World`" interactive example that just shows that the software is in place for TensorFlow can be run as follows:

Example

```
[root@bright90 ~]# module load shared
[root@bright90 ~]# module load tensorflow-py36-cuda10.1-gcc
[root@bright90 ~]# module load openmpi-geib-cuda10.1-gcc
[root@bright90 ~]# python3.6
Python 3.6.9 (default, Nov 20 2019, 21:00:07)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf

>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()

name: Tesla K40c major: 3 minor: 5 memoryClockRate(GHz): 0.745
pciBusID: 0000:00:08.0
totalMemory: 11.17GiB freeMemory: 11.09GiB

>>> sess.run(hello)
b'Hello, TensorFlow!'
>>> a = tf.constant(10)
>>> b = tf.constant(32)
>>> sess.run(a+b)
42
>>>
```

2.2 Convolutional Neural Network

The following trains a convolutional neural network similar to LeNet-5.

The code uses the TensorFlow convolutional module at `/cm/shared/apps/tensorflow-py36-cuda10.1-gcc/current/models/tutorials/image/mnist/convolutional.py`. It picks up training images and labels from the MNIST site, and places them in a directory `data` if it needs to. The images are then used to train and validate the model. End users would be expected to train the neural network within their home directories.

Example

```
[root@bright90 ~]# module load tensorflow-py36-cuda10.1-gcc
[root@bright90 ~]# module load openmpi-geib-cuda10.1-gcc
[root@bright90 ~]# cd /cm/shared/apps/tensorflow-py36-cuda10.1-gcc/current/models/tutorials/image/mnist/
[root@bright90 mnist]# python convolutional.py
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz

2018-05-02 13:37:08.796152: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1344]\
 Found device 0 with properties:
name: Tesla K40c major: 3 minor: 5 memoryClockRate(GHz): 0.745
pciBusID: 0000:00:08.0
totalMemory: 11.17GiB freeMemory: 11.09GiB
Initialized!

Step 0 (epoch 0.00), 25.9 ms
Minibatch loss: 8.334, learning rate: 0.010000
Minibatch error: 85.9%
Validation error: 84.6%
Step 100 (epoch 0.12), 158.8 ms
Minibatch loss: 3.267, learning rate: 0.010000
Minibatch error: 7.8%
Validation error: 7.5%
Step 200 (epoch 0.23), 154.9 ms
Minibatch loss: 3.355, learning rate: 0.010000
Minibatch error: 9.4%
...
Validation error: 0.8%
Step 8400 (epoch 9.77), 160.8 ms
Minibatch loss: 1.595, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 0.8%
Step 8500 (epoch 9.89), 157.6 ms
Minibatch loss: 1.611, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 0.9%
Test error: 0.7%
[root@bright90 ~]#
```

2.3 Image Recognition

The following session shows a pre-trained model running the `classify_image.py` script to recognize a test image of a panda:

Example

```
[root@bright90 ~]# module load tensorflow-py36-cuda10.1-gcc
[root@bright90 ~]# module load openmpi-geib-cuda10.1-gcc
[root@bright90 ~]# cd /cm/shared/apps/tensorflow-py36-cuda10.1-gcc/current/models/tutorials/image/imagenet
[root@bright90 imagenet]# python classify_image.py --image_file cropped_panda.jpg
2018-05-02 18:03:05.773697: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1344]\
 Found device 0 with properties:
name: Tesla K40c major: 3 minor: 5 memoryClockRate(GHz): 0.745
pciBusID: 0000:00:08.0
totalMemory: 11.17GiB freeMemory: 11.09GiB
>> Downloading inception-2015-12-05.tgz 100.0%
Successfully downloaded inception-2015-12-05.tgz 88931400 bytes.
2018-05-02 18:03:06.725257: W tensorflow/core/framework/op_def_util.cc:343] Op Batch\
NormWithGlobalNormalization is deprecated. It will cease to work in GraphDef version\
 9. Use tf.nn.batch_normalization().
giant panda, panda, panda bear, coon bear, Ailuropoda melanoleuca (score = 0.89107)
indri, indris, Indri indri, Indri brevicaudatus (score = 0.00779)
lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens (score = 0.00296)
custard apple (score = 0.00147)
earthstar (score = 0.00117)
[root@bright90 imagenet]#
```

2.4 Iris Classification

The iris flower data set is a standard raw data set, originally measured by R. A. Fisher in 1936. The data values in the set are a list of, per flower, the petal length and width, and the sepal length and width. Sepals are leaflike "under petals" under the actual petals. Based on these 4 attributes per flower, the fifth attribute—the species that the iris flower belongs to—can be determined. The determination of the species does not have an obvious solution because it depends on multiple variables at the same time. Fisher came up with a mathematical method to determine the species.

In machine learning, the iris flower data set is commonly used to illustrate concepts. An iris classification program, `premade_estimator`, is therefore a core TensorFlow program. The following session shows a run with `premade_estimator`, which downloads the data values used for training if required (some output ellipsized or removed for clarity):

```
[root@bright90 ~]# module load tensorflow-py36-cuda10.1-gcc
[root@bright90 ~]# module load openmpi-geib-cuda10.1-gcc
[root@bright90 ]# cd /cm/shared/apps/tensorflow-py36-cuda10.1-gcc/current/models/samples/core/get_started
[root@bright90 get_started]# python premade_estimator.py
Downloading data from http://download.tensorflow.org/data/iris_training.csv
16384/2194 [==============================================================- 0s 0us/step
Downloading data from http://download.tensorflow.org/data/iris_test.csv
16384/573 [==================================================- 0s 0us/step
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmpLqap1u
INFO:tensorflow:Using config: '_save_checkpoints_secs': 600, ' ...'_save_summary_steps': 100
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
```

```
2018-05-02 15:50:53.815952: I tensorflow/co...Found device 0 with properties:
name: Tesla K40c major: 3 minor: 5 memoryClockRate(GHz): 0.745
pciBusID: 0000:00:08.0
totalMemory: 11.17GiB freeMemory: 11.09GiB
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into /tmp/tmpLqap1u/model.ckpt.
INFO:tensorflow:loss = 192.68596, step = 1
INFO:tensorflow:global_step/sec: 468.658
INFO:tensorflow:loss = 12.971608, step = 101 (0.214 sec)
INFO:tensorflow:global_step/sec: 685.834
...
INFO:tensorflow:loss = 4.189755, step = 901 (0.158 sec)
INFO:tensorflow:Saving checkpoints for 1000 into /tmp/tmpLqap1u/model.ckpt.
INFO:tensorflow:Loss for final step: 6.7716765.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-05-02-13:50:58
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /tmp/tmpLqap1u/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-05-02-13:50:58
INFO:tensorflow:Saving dict for global step 1000: accuracy = 0.96666664,...loss = 1.6617917

Test set accuracy: 0.967

INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /tmp/tmpLqap1u/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.

Prediction is "Setosa" (99.9%), expected "Setosa"

Prediction is "Versicolor" (99.5%), expected "Versicolor"

Prediction is "Virginica" (97.7%), expected "Virginica"
```

More information on the program can be found at `https://www.tensorflow.org/tutorials/quickstart/beginner`.

3

Running PyTorch

This chapter goes through some example runs with PyTorch. Some output messages have been removed in the runs for readability.

It assumes PyTorch has been installed with a package manager from the Bright Cluster Manager repository. For example with:

```
yum install cm-pytorch-py36-cuda10.1-gcc
```

In addition, PyTorch requires an OpenMPI implementation to work. As described in Chapter 3 of the *User Manual*, the Bright repositories provide different OpenMPI packages. This allows the user to choose which one to use, depending on, for example, which interconnect is being used, or if CUDA support is required.

In this chapter, the `cm-openmpi-geib-cuda10.1-gcc` package is used.

3.1 Variational Autoencoders (VAEs)

The following example shows how to train *Variational Autoencoders*[1], powerful generative models that can be used for a wide variety of applications.

The source code is available in the official PyTorch GitHub repository and can be fetched as follows:

```
[root@bright90 ~]# git clone https://github.com/pytorch/examples.git
[root@bright90 ~]# cd examples/
[root@bright90 ~]# git checkout 6c51ca5a614cfdbdcd4e8c3e70321c5f6defb177
[root@bright90 ~]# cd vae/
```

The Variational Autoencoders network is trained by default for 10 epochs. The required dataset can automatically be downloaded and extracted with:

Example

```
[root@bright90 ~]# module load shared
[root@bright90 ~]# module load pytorch-py36-cuda10.1-gcc
[root@bright90 ~]# module load openmpi-geib-cuda10.1-gcc
[root@bright90 ~]# python3.6 main.py
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
../data/MNIST/raw/train-images-idx3-ubyte.gz
9920512it [00:04, 2041116.37it/s]
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw
...
Processing...
Done!
```

[1] Original paper: "`Auto-Encoding Variational Bayes`" by Diederik P Kingma and Max Welling: `https://arxiv.org/abs/1312.6114`.

```
Train Epoch: 1 [0/60000 (0%)] Loss: 550.187805
Train Epoch: 1 [1280/60000 (2%)] Loss: 323.104736
Train Epoch: 1 [2560/60000 (4%)] Loss: 237.460938
...
Train Epoch: 1 [58880/60000 (98%)] Loss: 130.540909
====> Epoch: 1 Average loss: 164.1742
====> Test set loss: 127.8219
Train Epoch: 2 [0/60000 (0%)] Loss: 127.949753
...
Train Epoch: 10 [58880/60000 (98%)] Loss: 107.980888
====> Epoch: 10 Average loss: 106.1472
====> Test set loss: 105.8715
```

The output sampled digits can be found in the `results` directory:

```
[root@bright90 ~]# ls results/
reconstruction_10.png  reconstruction_4.png  reconstruction_8.png  sample_2.png  sample_6.png
reconstruction_1.png   reconstruction_5.png  reconstruction_9.png  sample_3.png  sample_7.png
reconstruction_2.png   reconstruction_6.png  sample_10.png         sample_4.png  sample_8.png
reconstruction_3.png   reconstruction_7.png  sample_1.png          sample_5.png  sample_9.png
```

4

The Jupyter Notebook Environment Integration

4.1 Introduction

This chapter covers the installation and usage of the Jupyter Notebook environment in Bright Cluster Manager. An overview of the concepts and terminology follows.

What Is Jupyter Notebook?

Jupyter Notebook[1], or Jupyter, is a client-server open-source application that provides a convenient way for a cluster user to write and execute *notebook documents* in an interactive environment.

In Jupyter, a notebook document, or notebook, is content that can be managed by the application. Notebooks are organized in units called *cells* and can contain both executable code, as well as items that are not meant for execution.

Items not meant for execution can be, for example: explanatory text, figures, formulae, or tables. Notebooks can also store the inputs and outputs of an interactive session.

Notebooks can thus serve as a complete record of a user session, interleaving code with rich representations of resulting objects.

These documents are encoded as JSON files and saved with the `.ipynb` extension. Since JSON is a plain text format, notebooks can be version-controlled, shared with other users and exported to other formats, such as HTML, LaTeX, PDF, and slide shows.

What Is A Notebook Kernel?

A computational engine handling the various types of requests in a notebook (e.g. code execution, code completions, inspection) and providing replies to the user is called *notebook kernel*, or kernel. Usually kernels only allow execution of a single language, but kernels for many languages having varying quality and features are available.

What Is JupyterHub?

Jupyter on its own is single user. *JupyterHub*[2] allows it to provide a multi-user service, and is therefore commonly installed with Jupyter. JupyterHub is an open-source project that supports a number of authentication protocols and can be configured in order to provide access to a subset of users.

What Is JupyterLab?

JupyterLab[3] is a modern powerful interface for Jupyter, enabling users to work with notebooks and other applications, such as terminals or file browsers, in a flexible, integrated, and extensible manner. JupyterLab works out of the box with JupyterHub and can be used to arrange the user interface to support a

[1] https://jupyter-notebook.readthedocs.io/
[2] https://jupyterhub.readthedocs.io/
[3] https://jupyterlab.readthedocs.io/

wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is open-source and extensible with plugins that can customize or enhance any part of the interface, including themes, file editors, keyboard shortcuts, and other components.

What Is Jupyter Enterprise Gateway?

By default, Jupyter runs kernels locally, potentially exhausting server resources. *Jupyter Enterprise Gateway*[4] is a pluggable open-source framework that can be used to leverage local underlying resource managers, such as Slurm or Kubernetes, to distribute kernels across the compute cluster. In addition to scalability, it also provides an improved multi-user support and more granular security for Jupyter, making it suitable for enterprise, scientific, and academic implementations.

In Bright Cluster Manager, all the technologies mentioned in these sections are combined to provide a powerful, customizable and user-friendly Jupyter Notebook web interface running on a lightweight, multi-tenant, multi-language, scalable and secure environment, ready for a wide range of enterprise scenarios.

For convenience, in the following sections, *Jupyter* is used to collectively refer to Jupyter Notebook, JupyterHub, JupyterLab and Jupyter Enterprise Gateway.

4.2 Jupyter Notebook Environment Installation

Bright distributes Jupyter via two packages: `cm-jupyter` and `cm-jupyter-local`.

`cm-jupyter` is installed, as other machine learning packages are installed, in the `/cm/shared` directory, which is by default exported over NFS. As a result, Jupyter is available to all the compute nodes by default. `cm-jupyter` provides Jupyter Notebook, JupyterHub, JupyterLab and Jupyter Enterprise Gateway.

`cm-jupyter-local` provides the JupyterHub system service (`cm-jupyterhub.service`) and is therefore designed to be installed only on the node exposing users to the web login page for Jupyter. This node is typically the head node.

In addition to these two packages, Bright also distributes some custom Jupyter kernels. The custom kernels support different resource managers and languages via Jupyter Enterprise Gateway.

The `cm-jupyter-setup` script can be run on the head node of the cluster. It deploys a working Jupyter environment with minimal effort. The script comes with Bright Cluster Manager's `cm-setup` package.

It has no prerequisites, and can also be run before configuring any resource manager, such as Kubernetes or Slurm.

The `cm-jupyter-setup` script installs the two packages `cm-jupyter` and `cm-jupyter-local`. By default, the Jupyter environment initially contains only Jupyter's default Python 3 kernel, which runs on the head node. The script does not install any custom Jupyter kernels developed by Bright.

After having deployed Jupyter with the setup, an administrator can install, remove or customize kernels at any time (sections 4.2.3, 4.3, and 4.4.1).

4.2.1 Jupyter Setup

The default Jupyter architecture deployed by `cm-jupyter-setup` is shown in figure 4.1.

[4] https://jupyter-enterprise-gateway.readthedocs.io/

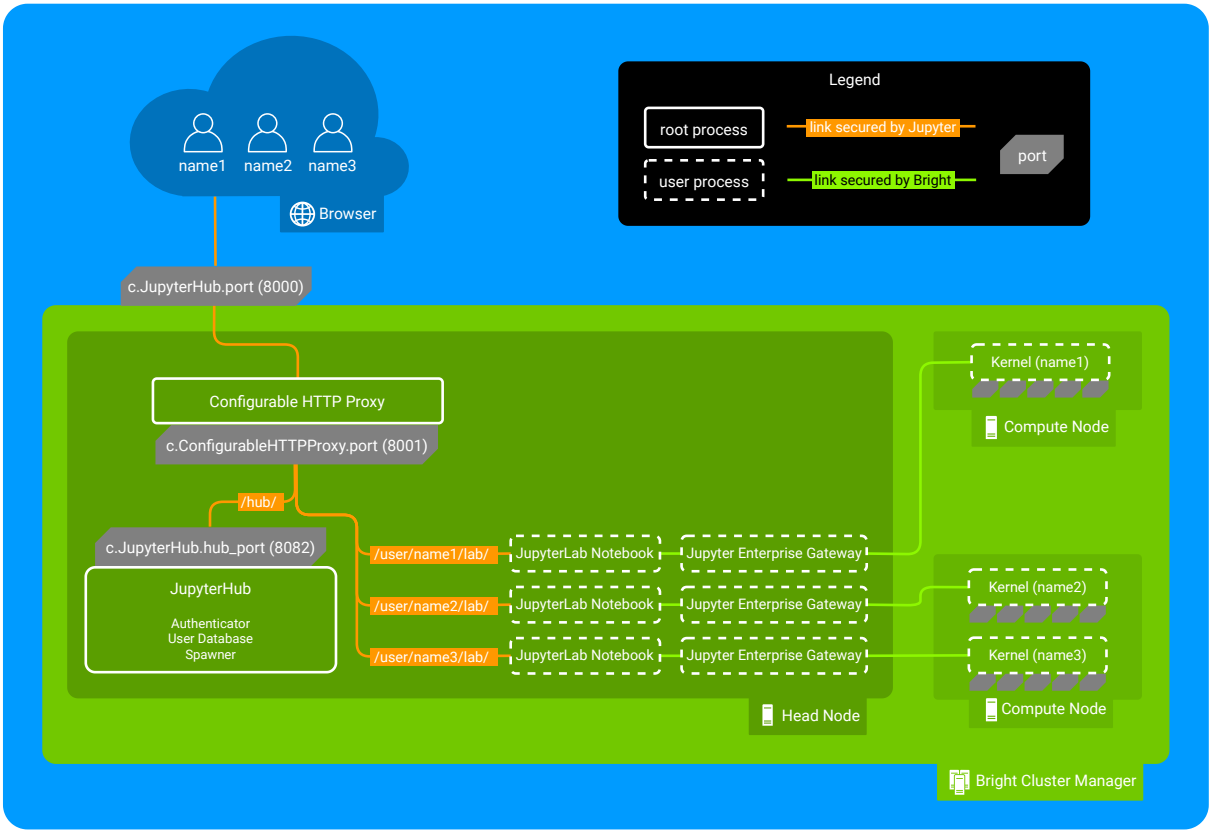

Figure 4.1: Jupyter architecture

In that architecture, the `cm-jupyterhub.service` provided by `cm-jupyter-local` starts JupyterHub on the port specified by `c.JupyterHub.hub_port` (default: 8082).

JupyterHub then automatically spawns a dynamic proxy to route HTTP requests via Bright's `cm-npm-configurable-http-proxy` package.

The proxy is the only process that listens for clients' requests on Jupyter's public interface, as specified by `c.JupyterHub.port` (default: 8000).

JupyterHub instructs the proxy on how requests should be dispatched by using its REST API. This is exposed at the port specified by `c.ConfigurableHTTPProxy.port` (default: 8001).

At this stage, users accessing Jupyter with their browsers are either redirected to the Hub (e.g. for authentication) or to their single-user servers to run notebooks.

By default, single-user servers are accessed at `/user/[username]`, while JupyterHub is accessed at `/hub`[5].

The `cm-jupyter-setup` script automatically installs JupyterLab and sets `/lab` as the default URL (`c.Spawner.default_url`) to redirect users to the new interface.

Finally, Jupyter Enterprise Gateway is integrated to spawn kernels on different nodes by configuring `jupyterhub.spawner.LocalProcessSpawner` as the default spawning mechanism (`c.JupyterHub.spawner_class`), and `jupyterhub-singleuser-gw` as the default spawning command (`c.Spawner.cmd`). When a new notebook is started, Jupyter Enterprise Gateway scans for an available port, and spawns a kernel chosen by the user on an appropriate cluster node. This process is then connected to JupyterLab.

JupyterHub and its HTTP proxy are run as two root processes, while JupyterLab and Jupyter Enterprise Gateway are run as user processes.

The privileges of the kernels spawned by Jupyter Enterprise Gateway can be configured by cluster

[5]https://jupyterhub.readthedocs.io/en/stable/reference/technical-overview.html

administrators, and depend on the underlying computational engine (e.g. Kubernetes). By default, all the kernels provided by Bright are run by user processes.

Communication between users' browsers, JupyterHub, its HTTP proxy, and JupyterLab, is secured by default by JupyterHub. On the other hand, communication between JupyterLab, Jupyter Enterprise Gateway, and kernels, is secured by Bright.

Administrators can customize their Jupyter integration by setting some of the aforementioned options when the `cm-jupyter-setup` script runs. New values are automatically handled by Bright Cluster Manager and written to Jupyter configuration files.

Other configuration options can be found in `/cm/shared/apps/jupyter/current/etc/jupyter/jupyterhub_config.py`.

4.2.2 Verifying Jupyter Installation

The `cm-jupyter-setup` script automatically starts the `cm-jupyterhub` service. It can take some time until the service is fully up and running, even if `systemctl status cm-jupyterhub -l` shows that the service is active.

Once functioning, the Jupyter web interface is accessible with a browser using the HTTPS protocol on the specified port (figure 4.2):

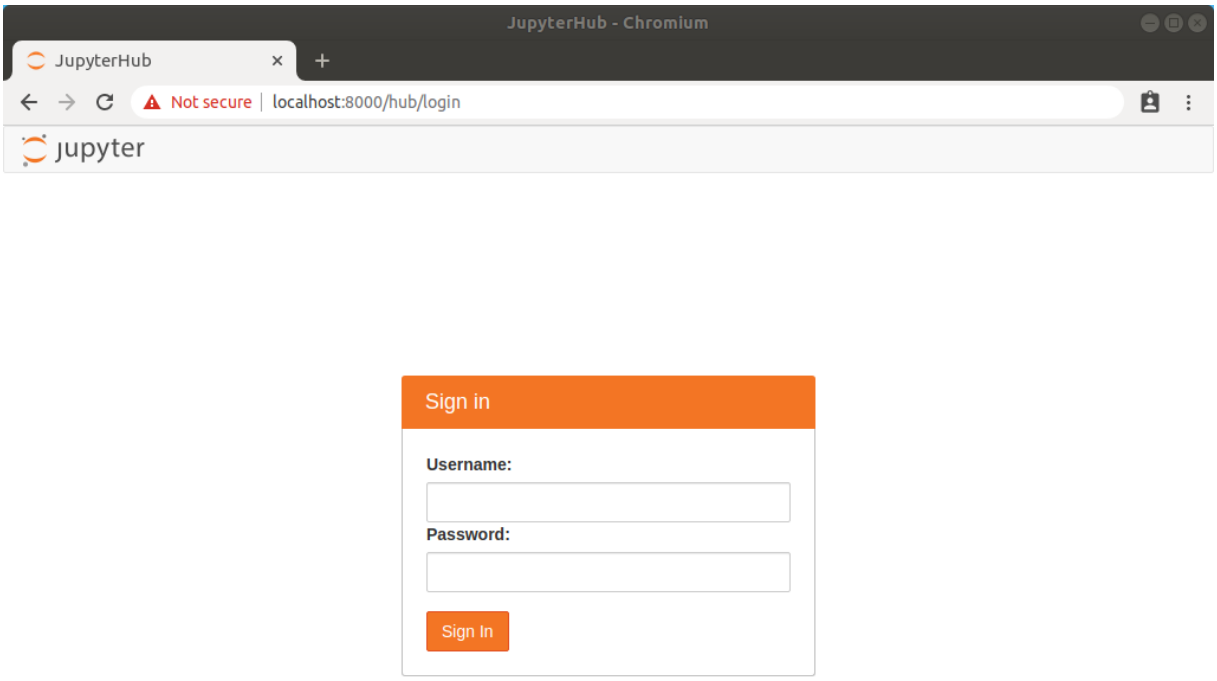

Figure 4.2: JupyterHub login screen

Any user registered in the Linux-PAM system can login to Jupyter.

For example, a test user `jupyterhubuser` with password `jupyterhubuser` can be created with:

Example

```
[root@bright90 ~]# cmsh -c "user; add jupyterhubuser; set password jupyterhubuser; commit"
```

After the first login, a new single-user server is spawned (figure 4.3):

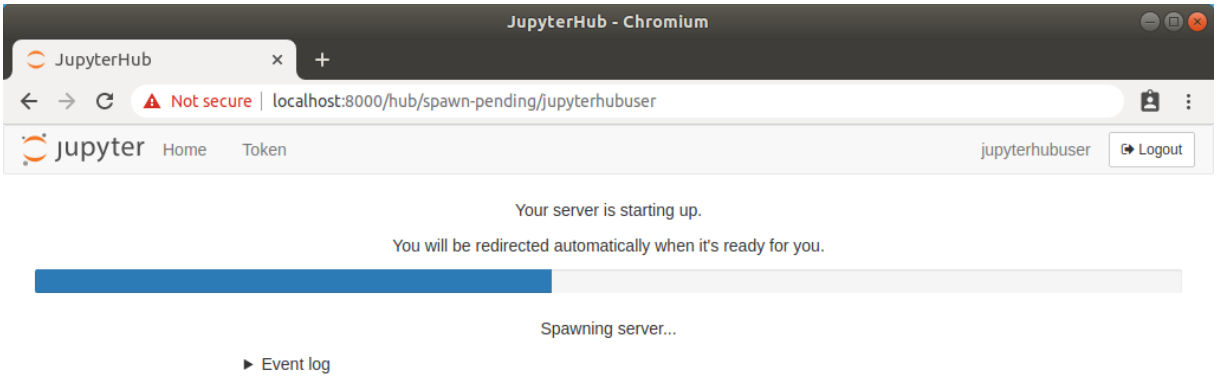

Figure 4.3: JupyterHub starting single-user server

Users are redirected to the JupyterLab interface and have access to Jupyter's default Python 3 kernel (figure 4.4):

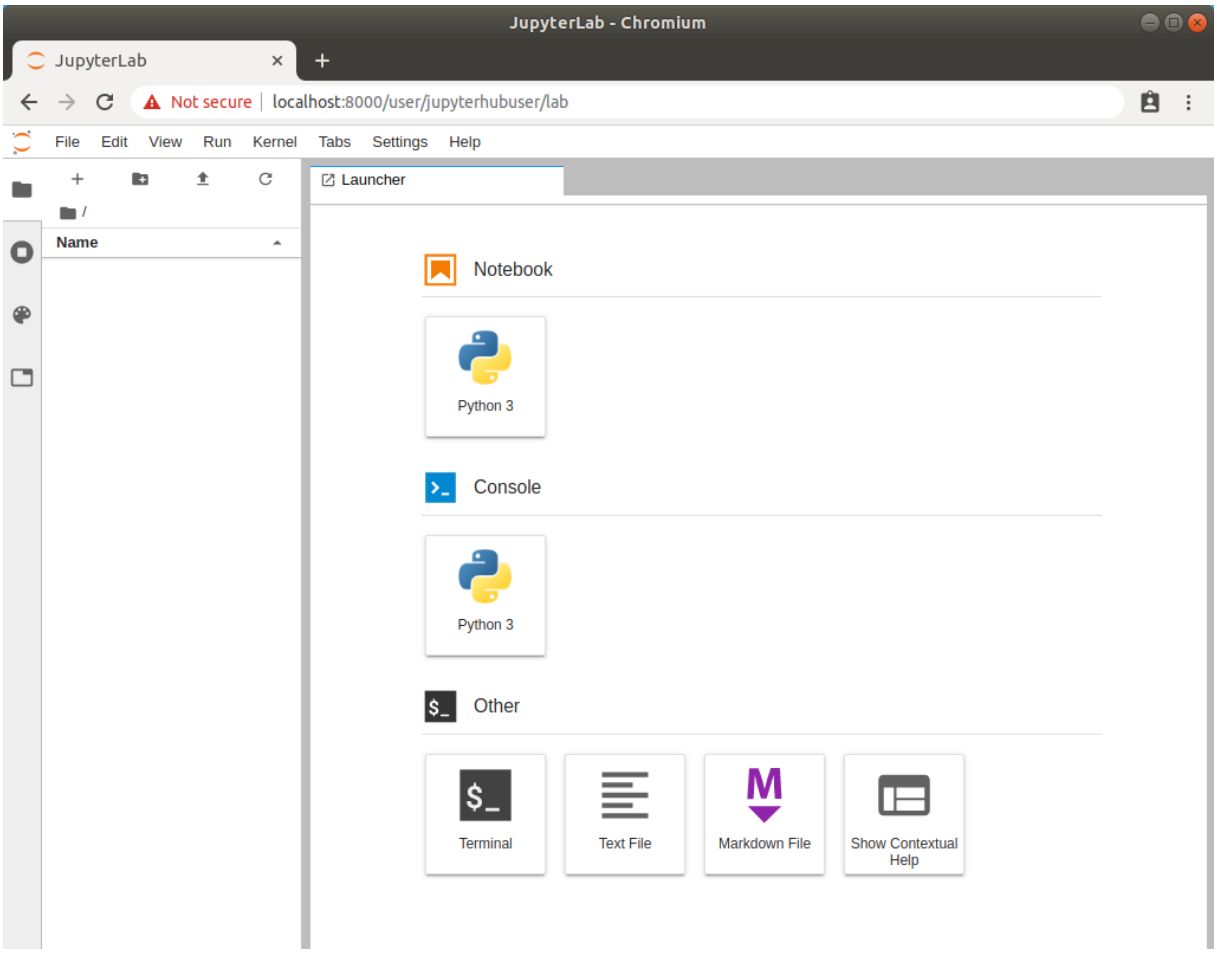

Figure 4.4: JupyterLab Launcher

4.2.3 Kernels Installation

Bright Cluster Manager includes some custom Jupyter kernels supporting different resource managers and languages via Jupyter Enterprise Gateway. For convenience, a summary of the available kernels and their requirements is shown in table 4.1:

Table 4.1: Available Jupyter kernels for Bright and their requirements

Package name	Requirement	Description
`cm-jupyter-eg-kernel-k8s`	Kubernetes[1]	Python 3.6 on Kubernetes
`cm-jupyter-eg-kernel-k8s-r`	Kubernetes[2]	R on Kubernetes
`cm-jupyter-eg-kernel-k8s-spark`	Kubernetes[3]	Python 3.7 + Spark 3.0 on Kubernetes
`cm-jupyter-eg-kernel-slurm-py36`	Slurm	Python 3.6 via Slurm
`cm-jupyter-eg-kernel-slurm-py37`	Slurm	Python 3.7 via Slurm

[1] Docker image: *brightcomputing/jupyter-kernel-sample:k8s-py36-1.0.0*
[2] Docker image: *brightcomputing/jupyter-kernel-sample:k8s-r-1.0.0*
[3] Docker image: *brightcomputing/jupyter-kernel-sample:k8s-spark-py37-1.0.0*
DockerHub kernels page: `https://hub.docker.com/r/brightcomputing/jupyter-kernel-sample/tags`

Jupyter kernel packages can be installed using their Linux distribution package manager in the usual way, as described in section 1.4.

Once installed, kernels are automatically added to Jupyter Launcher for all the users (figure 4.5):

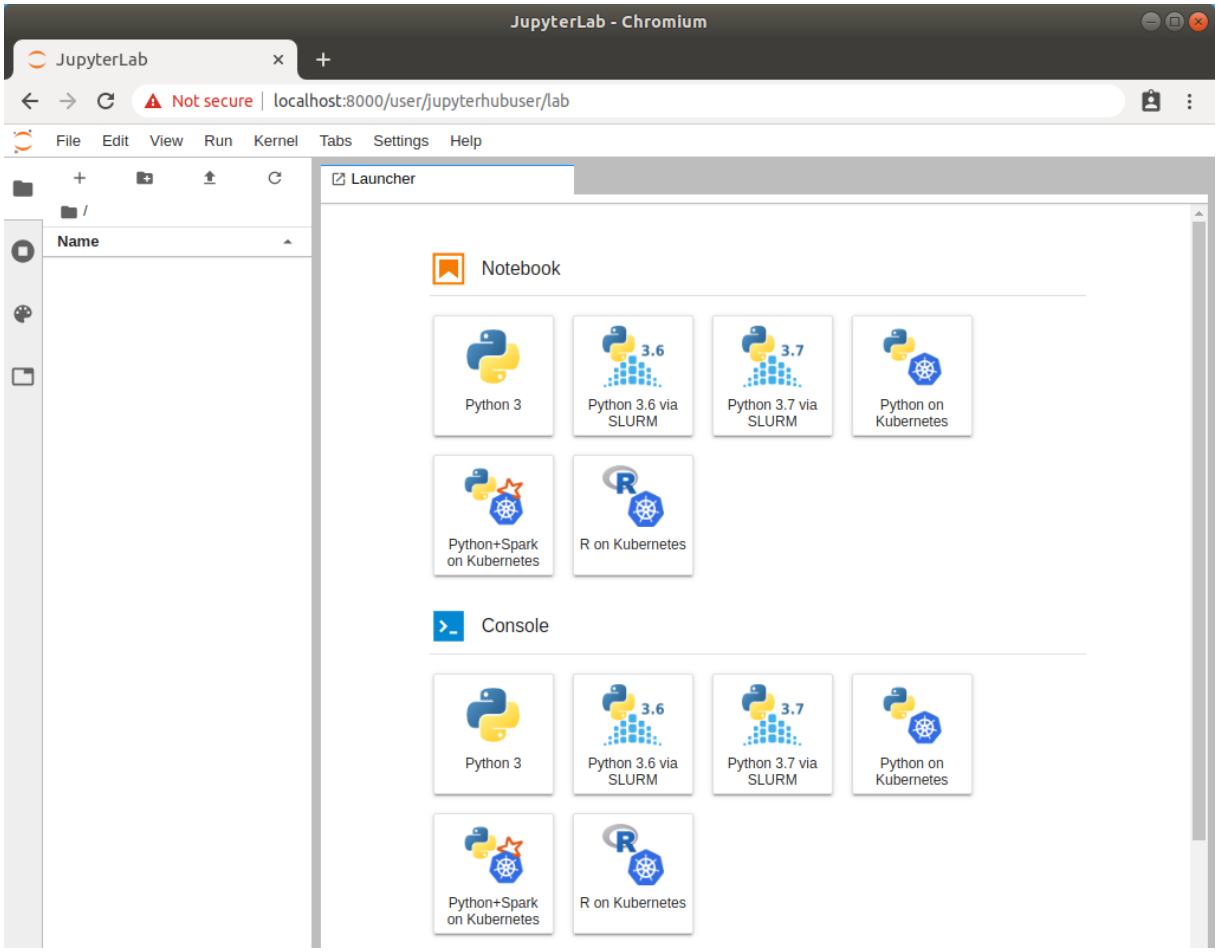

Figure 4.5: JupyterLab Launcher with Bright kernels

Running Jupyter Kernels On Compute Nodes With Slurm

Bright's `cm-jupyter-eg-kernel-slurm-py36` and `cm-jupyter-eg-kernel-slurm-py37` packages allow users to run Jupyter Enterprise Gateway kernels on cluster nodes via Slurm workload manager. These custom kernels offer a Python 3.6 and a Python 3.7 environment respectively.

In addition to installing the kernel package(s), an administrator has to make sure that Slurm is correctly configured on the cluster. Details on Slurm installation are provided in section 7.3 of the *Administrator Manual*.

The default configuration proposed by `cm-wlm-setup` in `Express` mode is usually sufficient to run Bright's Slurm kernels.

The node exposing the web login page for Jupyter to users has to be authorized to submit Slurm jobs. This is typically the case, since the JupyterHub login node is by default the head node, and `cm-wlm-setup` by default assigns the `slurmsubmit` role to the head node.

Finally, it is also responsibility of the administrator to make sure that relevant dependency packages are installed on the software image used by Slurm clients (typically the compute nodes). Missing packages may cause kernels to fail at startup or at run time.

In particular, administrators need to install `cm-python36` to use `cm-jupyter-eg-kernel-slurm-py36`, and `cm-python37` to use `cm-jupyter-eg-kernel-slurm-py37`.

Jupyter notebooks using Slurm kernels can natively use machine learning libraries and frameworks provided by Bright, if installed on the software image used by Slurm clients. Further details on machine learning packages installation on both the head node and the compute nodes are given in section 1.4.

Running Jupyter Kernels On Compute Nodes With Kubernetes

The following Bright packages allow users to run Jupyter Enterprise Gateway kernels on cluster nodes via Kubernetes:

Table 4.2: *Bright packages for running Jupyter Enterprise Gateway kernels on cluster nodes via Kubernetes*

Package	Description
`cm-jupyter-eg-kernel-k8s`	Custom kernel with Python 3.6
`cm-jupyter-eg-kernel-k8s-r`	Custom kernel with an R 3.4 environment
`cm-jupyter-eg-kernel-k8s-spark`	Custom kernel with Python 3.7 with Spark 3

It should be noted that the Spark environment provided by the `cm-jupyter-eg-kernel-k8s-spark` package in the preceding table is unrelated to the one available via `cm-spark-setup` (section 9.6.1 of the *Administrator Manual*). The two Spark instances may also differ in their major versions.

In addition to installing the kernel package(s), an administrator has to make sure that Kubernetes is correctly configured on the cluster. Details on Kubernetes installation are provided in section 9.3 of the *Administrator Manual*.

The default configuration proposed by `cm-kubernetes-setup` is usually sufficient to run Bright's Kubernetes kernels. However, it is the responsibility of the administrator to add users registered in the Linux-PAM system to Kubernetes.

For example, a test user `jupyterhubuser` can be added to Kubernetes with:

Example

```
[root@bright90 ~]# cm-kubernetes-setup --add-user jupyterhubuser
```

For every new user added, `cm-kubernetes-setup` automatically generates a dedicated namespace with a name in the form <*user*>-restricted. For instance, the command in the example above, creates the namespace: `jupyterhubuser-restricted`

By default, no privileges restriction is enforced on these dedicated namespaces. Cluster administrators are strongly urged to configure a pods security policy. Further details on this topic are provided in section 9.3.8 of the *Administrator Manual*.

In order to speed up kernel creation for the first users logging into to JupyterLab, Bright recommends that all the relevant Kubernetes images are pre-loaded on the cluster nodes that the Jupyter Enterprise Gateway contacts.

Jupyter notebooks using Kubernetes kernels cannot natively use machine learning libraries and frameworks provided by Bright, even if installed on the software image used by Kubernetes.

4.3 Kernels Customization

Any user registered in the Linux-PAM system can list the available Jupyter kernels via the command line.

The following example is executed in an environment containing Bright's `cm-jupyter-eg-kernel-slurm-py37` kernel and Jupyter's default Python 3 kernel:

Example

```
[jupyterhubuser@bright90 ~]# module load jupyter
Loading jupyter/6.0.0
  Loading requirement: python37 cm-npm-configurable-http-proxy/4.2.0
[jupyterhubuser@bright90 ~]# jupyter kernelspec list
Available kernels:
  jupyter-eg-kernel-slurm-py37   /cm/shared/apps/jupyter/6.0.0/share/jupyter/kernels/
                                 jupyter-eg-kernel-slurm-py37
  python3                        /cm/shared/apps/jupyter/6.0.0/share/jupyter/kernels/python3
```

Every listed directory contains a `kernel.json` file describing how Jupyter Enterprise Gateway will spawn and communicate with kernels:

Example

```
[jupyterhubuser@bright90 ~]# ls /cm/shared/apps/jupyter/current/share/jupyter/kernels/*/kernel.json
/cm/shared/apps/jupyter/current/share/jupyter/kernels/jupyter-eg-kernel-slurm-py37/kernel.json
/cm/shared/apps/jupyter/current/share/jupyter/kernels/python3/kernel.json
```

Administrators can customize `kernel.json` files in the `/cm/shared` directory to offer new features to all Jupyter users.

In a similar way, a user can create a local copy of a kernel specification in the home directory and customize it. By default, the Jupyter data directory for a user is located at `$HOME/.local/share/jupyter`.

This path can be verified with Jupyter with the `--paths` option:

Example

```
[jupyterhubuser@bright90 ~]# jupyter --paths
config:
    /home/jupyterhubuser/.jupyter
    /cm/local/apps/python37/etc/jupyter
    /cm/shared/apps/jupyter/6.0.0/etc/jupyter
    /etc/jupyter
data:
    /home/jupyterhubuser/.local/share/jupyter
    /cm/local/apps/python37/share/jupyter
    /cm/shared/apps/jupyter/6.0.0/share/jupyter
    /usr/share/jupyter
runtime:
    /home/jupyterhubuser/.local/share/jupyter/runtime
[jupyterhubuser@bright90 ~]# cp -R
/cm/shared/apps/jupyter/current/share/jupyter/kernels/\
jupyter-eg-kernel-slurm-py37 /home/jupyterhubuser/.local/share/jupyter
[jupyterhubuser@bright90 ~]# # edit /home/jupyterhubuser/\
.local/share/jupyter/jupyter-eg-kernel-slurm-py37/kernel.json
```

This section provides information on how Bright's kernel specifications can be customized to achieve different goals. Further details can be found in the official Jupyter Enterprise Gateway documentation on kernel launchers[6] and the remote kernel manager[7].

[6] https://jupyter-enterprise-gateway.readthedocs.io/en/latest/system-architecture.html#kernel-launchers

[7] https://jupyter-enterprise-gateway.readthedocs.io/en/latest/system-architecture.html#remote-kernel-manager

4.3.1 Slurm Kernel—Customize Available Modules For Jobs

Modules available for Slurm jobs submitted via Bright's `cm-jupyter-eg-kernel-slurm-py36` and `cm-jupyter-eg-kernel-slurm-py37` are defined in the `metadata→process_proxy→config→environ` entry of the kernel.json file:

```
{
  "display_name": "Python 3.7 via SLURM",
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        "environ": [
            "source /etc/profile.d/modules.sh",
            "module load shared slurm jupyter-eg-kernel-slurm-py37"
        ],
        [...]
      }
    }
  },
  [...]
}
```

Any module available on the cluster can be added to the kernel definition and loaded by Slurm jobs. For example, TensorFlow and MPI modules can be added as follows:

```
{
  "display_name": "Python 3.7 via SLURM",
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        "environ": [
            "source /etc/profile.d/modules.sh",
            "module load shared slurm jupyter-eg-kernel-slurm-py37 tensorflow-py37-cuda10.1-gcc \
openmpi-geib-cuda10.1-gcc"
        ],
        [...]
      }
    }
  },
  [...]
}
```

4.3.2 Slurm Kernel—Increasing The Number Of Concurrent Running Notebooks

By default, kernels spawned for Slurm jobs that are submitted via Bright's `cm-jupyter-eg-kernel-slurm-py36` and `cm-jupyter-eg-kernel-slurm-py37` Jupyter Enterprise Gateway occupy one node. This means that, by default, the number of concurrent Jupyter notebooks that can be run with Slurm depends on the number of compute nodes available for the queue (named `defq` by default).

A cluster administrator can however enable the Slurm setting `oversubscribe` to allow each node to run multiple kernels. The downside of doing this is a context-switching penalty. Analyzing the trade-offs for setting `oversubscribe` is beyond the scope of this section.

The oversubscription setting to allow up to 8 jobs to share the same nodes in Slurm for defq can be carried out as follows:

Example

```
[root@bright90 ~]# cmsh -c "wlm use slurm ; jobqueue ; use defq ; set oversubscribe YES:8 ; commit"
```

For Jupyter to use this configuration, the way jobs are submitted by Jupyter Enterprise Gateway has to be updated.

By default, the command to submit Slurm jobs via Bright's cm-jupyter-eg-kernel-slurm-py36 and cm-jupyter-eg-kernel-slurm-py37 is defined in the metadata→process_proxy→config→submit entry of the kernel.json file:

```
{
  "display_name": "Python 3.7 via SLURM",
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        "submit": ["sbatch --parsable"],
      }
    }
  },
  [...]
}
```

To enable Slurm oversubscription, the additional --oversubscribe flag has to be added:

```
{
  "display_name": "Python 3.7 via SLURM",
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        "submit": ["sbatch --parsable --oversubscribe"],
      }
    }
  },
  [...]
}
```

Details on Slurm configuration for GPUs are described in Section 3.13.2 of the *Administrator Manual*

4.3.3 Slurm Kernel—Use Of Different Slurm Deployments

Cluster administrators can deploy different Slurm instances via cm-wlm-setup. It is possible to configure Jupyter to use a specific instance by updating the modules available for jobs submitted via Bright's cm-jupyter-eg-kernel-slurm-py36 and cm-jupyter-eg-kernel-slurm-py37.

By default, the slurm module typically associated to the basic Slurm deployment is defined in the metadata→process_proxy→config→environ entry of the kernel.json file:

```
{
  "display_name": "Python 3.7 via SLURM",
```

```
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        "environ": [
            "source /etc/profile.d/modules.sh",
            "module load shared slurm jupyter-eg-kernel-slurm-py37"
        ],
        [...]
      }
    }
  },
  [...]
}
```

Any Slurm module that is available on the cluster can be used in the kernel definition, and can replace the default Slurm instance. For example, a `second-wlm-deployment` Slurm module can be used as follows:

```
{
  "display_name": "Python 3.7 via SLURM",
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        "environ": [
            "source /etc/profile.d/modules.sh",
            "module load shared slurm/second-wlm-deployment jupyter-eg-kernel-slurm-py37"
        ],
        [...]
      }
    }
  },
  [...]
}
```

4.3.4 Kubernetes Kernel—Use Of Different Pod Images

To run Kubernetes kernels, cluster administrators can offer Jupyter users container images that are different to the ones provided by default by Bright.

By default, Kubernetes kernels use the image defined in the kernel.json file, in the `metadata→process_proxy→config→image` entry:

```
{
  "display_name": "Python on Kubernetes",
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        [...]
        "image": "brightcomputing/jupyter-kernel-sample:k8s-py36-1.0.0",
        [...]
```

```
      }
    }
  },
  [...]
}
```

Bright does not provide support for third-party images. It is the responsibility of cluster administrators or users to make sure new container images are compatible with the Jupyter deployment. In particular, it is important for new images to provide a way to allow Jupyter Enterprise Gateway to connect and communicate with running pods.

In Bright's `cm-jupyter-eg-kernel-k8s`, `cm-jupyter-eg-kernel-k8s-r` and `cm-jupyter-eg-kernel-k8s-spark` packages (table 4.2), the communication with pods is established with the command specified in the `argv` entry of the kernel.json file:

```
{
  "display_name": "Python on Kubernetes",
  [...],
  "argv": [
    "/usr/bin/python3",
    "/ipykernel-k8s.py",
    "{kernel_id}"
  ]
}
```

or:

```
{
  "display_name": "R on Kubernetes",
  [...],
  "argv": [
    "/usr/bin/Rscript",
    "/irkernel-k8s.r",
    "{kernel_id}"
  ]
}
```

The corresponding Kubernetes images respectively include the `/ipykernel-k8s.py` and `/irkernel-k8s.r` scripts, so that Jupyter Enterprise Gateway requests can be handled.

These scripts, along with the `Dockerfiles` used to define Kubernetes images, provided by Bright, can be inspected on clusters under the appropriate `contrib` directories:

```
[root@bright90 ~]# ls /cm/shared/apps/jupyter-eg-kernel-k8s/current/contrib/py36-image/
Dockerfile  files
[root@bright90 ~]# ls /cm/shared/apps/jupyter-eg-kernel-k8s-spark/current/contrib/pyspark-image/
build.sh  Dockerfile  files
[root@bright90 ~]# ls /cm/shared/apps/jupyter-eg-kernel-k8s-r/current/contrib/r-image/
Dockerfile  files
```

In order to allow containers to be correctly started and stopped by Jupyter Enterprise Gateway, administrators or users have to include in their images a kernel script suitable for the target programming language. For example, for a Docker image designed to run a Python kernel, the `ipykernel-k8s.py` script must be added. This file is shipped with the `cm-jupyter-eg-kernel-k8s` package:

```
[root@bright90 ~]# ls /cm/shared/apps/jupyter-eg-kernel-k8s/current/contrib/py36-image/files/
ipykernel-k8s.py  requirements.py3.txt
```

Additional requirements for the kernel scripts may be defined, as suggested by the file associated in the previous example (i.e. `requirements.py3.txt`). These requirements must be met in the container images.

The following is an example of a minimal `Dockerfile` definition to run a Python kernel via Kubernetes:

```
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y python3.6 curl gcc python3.6-dev python3-pip python3-distutils
COPY files/requirements.py3.txt /tmp/requirements.py3.txt
RUN pip3 install -r /tmp/requirements.py3.txt
COPY files/ipykernel-k8s.py /ipykernel-k8s.py
```

The previous instructions can be also used to customize and run NGC Containers provided by NVIDIA. The following example shows how to customize a TensorFlow image provided by NVIDIA to define a `Dockerfile` compatible with Bright's Jupyter deployment:

```
FROM nvcr.io/nvidia/tensorflow:20.01-tf1-py3
COPY files/requirements.py36.txt /tmp/requirements.py36.txt
RUN pip3.6 install -r /tmp/requirements.py36.txt
COPY files/ipykernel-k8s.py /ipykernel-k8s.py
```

4.3.5 Kubernetes Kernel—Customizing Pod Configurations

Cluster administrators can customize pod configurations for Kubernetes kernels. Customizations may involve a wide range of options. These could be, among others: available resources, security restrictions, mounted volumes, and environment variables.

However, it is outside of the scope of this section to provide guidance on how to customize Kubernetes pods to achieve particular results.

In Bright's `cm-jupyter-eg-kernel-k8s`, `cm-jupyter-eg-kernel-k8s-r` and `cm-jupyter-eg-kernel-k8s-spark` packages, the pod configuration templates are defined within the kernel.json file under `metadata`→`process_proxy`→`config`→<*pod configuration key*>, where <*pod configuration key*> can be:

- `pod_template`
- `service_template`
- `secret_template`
- `config_map_template`

The file section looks like:

```
{
  "display_name": "Python on Kubernetes",
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        "pod_template":         "templates/pod.yaml.j2",
        "service_template":     "templates/service.yaml.j2",
        "secret_template":      "templates/secret.yaml.j2",
        "config_map_template":  "templates/config_map.yaml.j2",
        [...]
      }
    }
  },
```

```
  [...]
}
```

Pod templates can be found in the same directory of the kernel.json file.

For example, for Bright's `cm-jupyter-eg-kernel-k8s` package, the following file values are provided:

Example

```
[root@bright90 ~]# cd /cm/shared/apps/jupyter/6.1.0/share/jupyter/kernels/jupyter-eg-kernel-k8s/
[root@bright90 jupyter-eg-kernel-k8s]# ls
kernel.json  logo-64x64.png  templates
[root@bright90 jupyter-eg-kernel-k8s]# ls templates
config_map.yaml.j2  pod.yaml.j2  secret.yaml.j2  service.yaml.j2
```

4.3.6 Kubernetes Kernel—Use Of Different Kubernetes Deployments

Cluster administrators can deploy different Kubernetes instances via `cm-kubernetes-setup`. Jupyter can be made to use a specific instance by updating the modules available for jobs submitted via Bright's `cm-jupyter-eg-kernel-k8s`, `cm-jupyter-eg-kernel-k8s-r` and `cm-jupyter-eg-kernel-k8s-spark`.

By default, the `kubernetes` module associated with the Kubernetes deployment is defined in the `metadata→process_proxy→config→k8s_env_module` entry of the kernel.json file:

```
{
  "display_name": "Python on Kubernetes",
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        [...]
        "k8s_env_module": "kubernetes/default",
        [...]
      }
    }
  },
  [...]
}
```

If no module is specified, then the latest deployed instance is used as fallback. This is usually `kubernetes/default`.

Any Kubernetes module that is available on the cluster can be used in the kernel definition, and can replace the default Kubernetes instance. For example, a `second-k8s-deployment` Kubernetes module can be used as follows:

```
{
  "display_name": "Python on Kubernetes",
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        [...]
        "k8s_env_module": "kubernetes/second-k8s-deployment",
        [...]
      }
    }
```

```
  },
  [...]
}
```

4.4 Jupyter Notebook Environment Removal

Before removing Jupyter, the administrator should ensure that all kernels have been halted, and that no user is still logged onto the web interface. Stopping the `cm-jupyterhub` service with users that are still logged in, or with running kernels, has undefined behavior.

To remove JupyterHub, the script `cm-jupyterhub-setup` must be run, either in interactive mode, or with the option `--remove`.

Removing JupyterHub does not remove or affect Kurbernetes or Slurm deployments.

In any case, for a complete cleanup, the following packages must be removed: `cm-jupyter`, `cm-jupyter-local`, `cm-npm-configurable-http-proxy`, and every Bright Jupyter kernel.

4.4.1 Kernels Removal

Jupyter kernel packages distributed by Bright can be removed by the package manager for the Linux distribution in the usual way (section 1.5), similar to how other machine learning packages are removed.