Orchestrating a brighter world NEC

発行番号：OUCMC-super-技術-006-14

国立大学法人 大阪大学

D3 センター　御中

利用者向けマニュアル

日本電気株式会社

コンピュート統括部

改版履歴

版数	年月日	該当項目	概要説明	承認	査閲	作成
1.0	2021/05/06	全項目	新規作成	NEC	NEC	NEC
2.0	2021/05/26	1.2 1.4 1.5 3.1.3 3.3.4. 4.1.3. 5.2.2. 6.2.2. 6.2.4.	1.2 各環境の SW 構成の説明追加 1.4 各環境の SW 構成の説明追加 1.5 システム利用環境の説明追加 3.1.3 module コマンド実行例を加筆 3.3.4. jupyter コンテナのカスタマイズ手順を追加 4.1.3. ddt、Parallel Debugger の起動方法修正 5.2.2. ベクトル計算環境向けジョブクラスの DBG、INTV の上限変更 6.2.2. Gaussian の利用方法を追加 6.2.4.Gnuplot の利用方法を追加	NEC	NEC	NEC
3.0	2021/06/04	3.1.3 5.1.3 3.3.1 3.3.2 3.3.5 4.1.1 4.2.4 4.3.4 4.4.6 4.5 4.7 6.2.5	3.1.3 5.1.3 modules.sh を使用する記述の削除 3.3.1 3.3.2 3.3.5 ホームから任意のディレクトリで操作手順に変更。停止手順を追加 4.1.1 その他のプログラム言語環境の起動方法を加筆 4.2.4 4.3.4 4.4.6 各環境のライブラリ利用方法を加筆 4.5 GNU コンパイラ利用方法を追加 4.7 python の利用方法を追加 6.2.5 GROMACS の利用方法のサンプルスクリプト修正	NEC	NEC	NEC
4.0	2021/06/29	5.1.3 5.1.4	5.1.3 (2) ジョブ管理システムへのバッチジョブ要求の加筆 5.1.4 (5) (6)バッチジョブの削除、	NEC	NEC	NEC

			キューの状態確認を追加			
		6.1	6.1 アプリケーション一覧を分類			
		6.2	6.2 ISV アプリケーションの利用方法を追加			
		6.3	6.3 OSS アプリケーションの利用方法を追加			
		6.4	6.4 国のプロジェクトのアプリケーション利用方法を追加			
5.0	2021/8/16	1.5.4 4.4.6 4.6 4.7 4.7.4 5.6 6.3.3 6.3.5 6.3.10	1.5.4 コンテナに関する説明を拡充 4.4.6 GPGPU 計算環境むけ HDF5 ライブラリの利用方法見直し 4.6 コンテナ利用の流れを見直し。sandbox を使用しないイメージ取得方法を記載。NGC を追加 4.7 GPU 対応 Python3 の利用方法を追加 4.7.4 ライブラリの利用(TensorFlow,Keras,Pytorch)を追加 5.6 コンテナの実行方法を追加 6.3.3 GAMESS の利用方法を追加 6.3.5 GROMACS の利用方法を追加 6.3.10 QuantumESPRESSO を追加	NEC	NEC	NEC
6.0	2021/9/28	5.4.3 5.4.4 6.3.10 5.6.1. 6.1 6.3.11 7.2	5.4.3 5.4.4 6.3.10 NEC MPI の mpirun 実行時に venode オプションを追加 5.6.1 ホスト OS 上の a.out を実行する手順の誤植を修正 6.1 アプリケーション一覧に PHASE/0 を追加 6.3.11 PHASE/0 の利用方法を追加 7.2 バケット操作の見直しに伴う改訂 ・s3dsbucket コマンドを追加 ・バケットの作成/同期手順を追加	NEC	NEC	NEC

7.0	2021/11/23	4.6.2	4.6.2 sandbox の作成と削除に関する改訂 ・(4)sandbox の削除を追加	NEC	NEC	NEC
8.0	2021/12/13	1.1 1.3.1 1.3.2 7.4 8.1.1.	ONION 拡張 PJ の変更内容を記載	NEC	NEC	NEC
9.0	2022/05/18	5 5.3.5 5.5.5 8.2.1 8.2.2 8.2.3	ジョブスクリプトで経過時間を指定しない場合、1 時間に変更 5.3.5 プロセス数のマニュアル指定方法を追加 5.5.5 RANK 毎の CPU、GPU の割当て方法を記載 8.2.1 8.2.2 ホーム画面画像の差し替え 8.2.3 ノード稼働状況の追加	NEC	NEC	NEC
10.0	2023/4/19	3.2.1 3.3.6 5.3.5 6.2.2 6.3.12	NICE DCV クライアントの入手先 URL を最新版へ修正 julia、R カーネルインストール済み jupyter コンテナの利用方法を追加 VTune プロファイラの使用例を追加 サンプルスクリプト内の誤記の修正 Paraview の利用方法修正	NEC	NEC	NEC
11.0	2023/10/12	1.2 1.4 3.3.1 3.3.6	汎用 CPU 計算環境、ベクトル計算環境、GPGPU 計算環境ノード、HPC フロントエンド、HPDA フロントエンドの OS の更新 推奨コンテナイメージ変更に伴う利用手順の修正	NEC	NEC	NEC
12.0	2025/4/10	6.3.7 4.2 4.3	ArmForge を LinaroForge に変更 Anaconda に関する項目を削除 miniforge の利用方法を追記 InteloneAPI 2025.0.1 の利用方法を追記	NEC	NEC	NEC

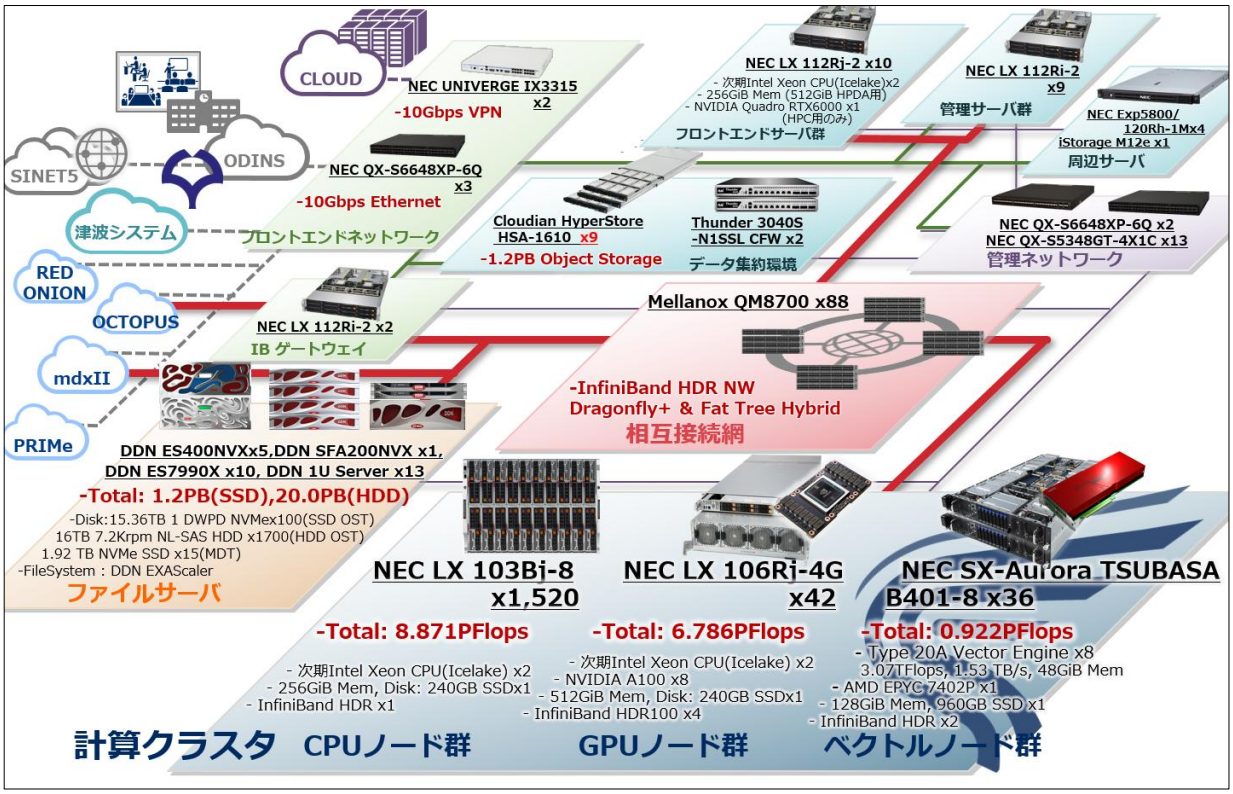
		全体	IntelMPI 2021.14 の利用方法を追記 ロードモジュールのバージョンを修正			
13.0	2026/03/05	8.1.1	Cloudian のアクセス URL を変更	NEC	NEC	NEC
14.0	2026/04/10	5.1.5	NQSV における Core ファイル出力設定を追記	NEC	NEC	NEC

目次

1. システム概要

1.1. システム全体図

「SQUID（Supercomputer for Quest to Unsolved Interdisciplinary Datascience）」は、汎用 CPU ノード群、ベクトルノード群、GPU ノード群、ファイルサーバから構成され、総理論演算性能 16.591 PFLOPS を有するスーパーコンピュータシステムです。



総演算性能	16.591PFLOPS	
ノード構成	汎用 CPU ノード群 1520 ノード（8.871 PFlops）	Intel® Xeon® Platinum 8368 (2.4GHz/38core) x2
	ベクトルノード群 36 ノード（0.922PFlops）	Type20A Vector Engine (3.07 TFlops,1.53TB/s) x8 AMD EPYC™ 7402P (2.8GHz/24core) x1
	GPU ノード群 42 ノード（6.797PFlops）	Intel® Xeon® Platinum 8368 (2.4GHz/38core) x2 NVIDIA A100 GPU(9.7TFlops) x8
相互接続網	InfiniBand HDR (200 Gbps)	
ファイルサーバ	DDN EXAScaler (SSD 領域 1.2 PB / HDD 領域 20PB)	

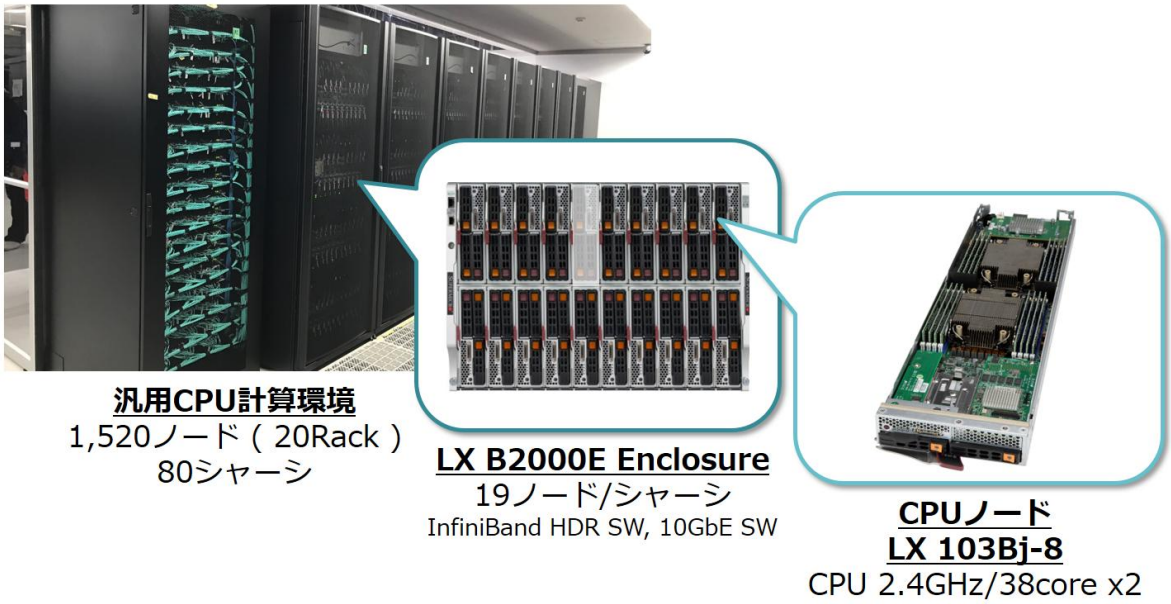
1.2. 3 つの計算環境

SQUID では、計算ノードの種別毎に 3 つの計算環境が利用可能です。以下が特徴になります。

計算環境	汎用 CPU 計算環境	ベクトル計算環境	GPGPU 計算環境
用途	一般的な計算用途。 大規模並列実行用	ベクトルエンジンによる 高メモリ帯域ベクトル演算用	GPGPU 加速器による 高速演算計算用
ノード数	1,520 ノード	288 VE	42 ノード
演算器	Intel Xeon Platinum 8368 (2.4GHz/38Core)x2	Type 20A Vector Engine (3.07 TFlops,1.53TB/s) x8	Intel Xeon Platinum 8368 (2.4GHz/38Core)x2
加速器	-	-	NVIDIA A100 x8
メモリ容量	256GiB	48GiB (HBM2)	512GiB
相互接続網	InfiniBand HDR x1	InfiniBand HDR x2	InfiniBand HDR100 x4

1.2.1. 汎用 CPU 計算環境

汎用 CPU 計算環境の汎用 CPU ノードは、「NEC LX 103Bj-8」1,520 ノードで構成されています。ブレードシステムのサーバで、ラックあたり 4 シャーシ、シャーシあたり 19 ノードで構成されます。



・**汎用 CPU ノード**

項目		構成
総ノード数		1,520 台
サーバ構成	プロセッサ	Intel Xeon Platinum 8368 (2.4 GHz/38core)　x2
	メモリ構成	256 GiB (16GiB DDR4-3200 RDIMM x16)
	ハードディスク	240GB SATA SSD x1

	インタフェース	InfiniBand HDR x1、25/10GbE x2、BMCx1

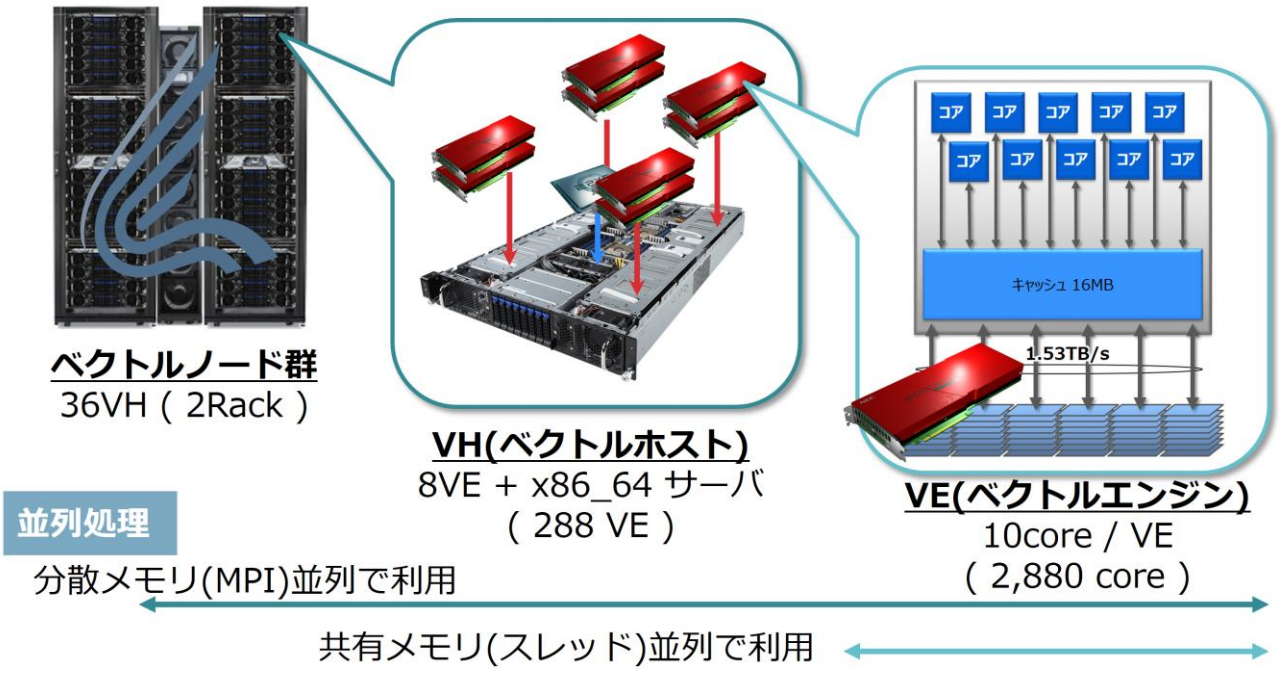
・ソフトウェア環境

項目	構成
OS	Rocky Linux 8.10 (64bit)
コンパイラ	Intel Parallel Studio Intel oneAPI
MPI	Intel MPI

ソフトウェア環境としては、Intel 社製プロセッサ向け最適化に優れた C/C++/FORTRANs の開発環境である「Intel Parallel Studio / Intel oneAPI」が利用可能です。同環境に同梱されている「Intel MPI」を標準 MPI としています。

1.2.2. ベクトル計算環境

ベクトル計算環境のベクトルノードは、「NEC SX-Aurora TSUBASA B401-8」288 VE(ベクトルエンジン)で構成されています。x86_64 サーバである VH(ベクトルホスト)に 8VE を搭載し、全 36VH で構成されます。



・ベクトルエンジン

項目		構成
総 VE 数		288 VE (1VH 当たり 8VE)
VE 構成	モデル名	Type 20A
	演算性能（倍精度）	307 GFlops / core　x 10 core
	メモリ	48 GiB HBM2

・**ベクトルホスト**

項目		構成
総 VH 数		36 台
VH 構成	プロセッサ	AMD EPYC 7402P Processor(2.8GHz/24core) x1
	メモリ構成	128 GiB
	インタフェース	InfiniBand HDR x2, 1000BASE-T x1, BMCx1

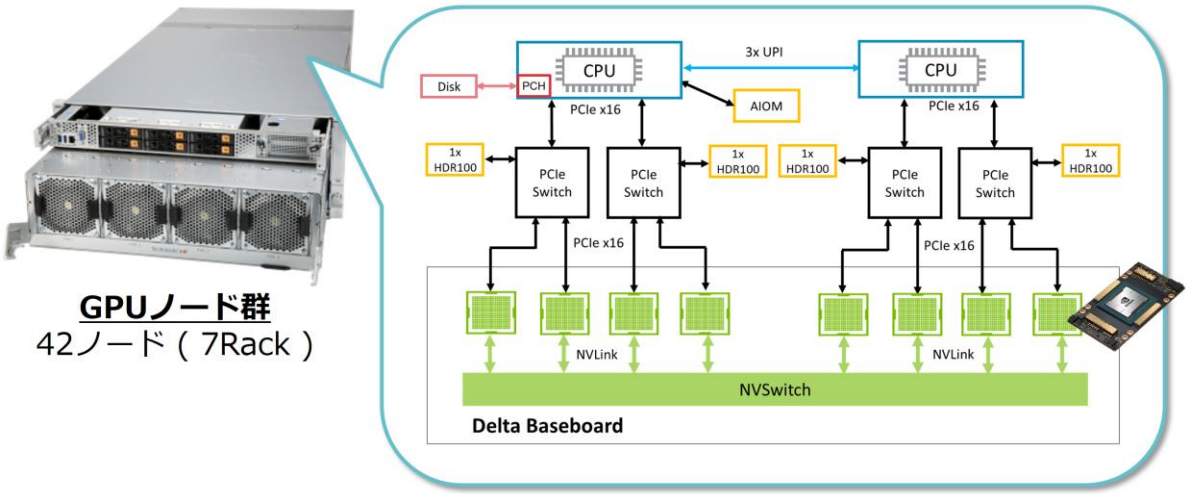
・**ソフトウェア環境**

項目	構成
OS	Rocky Linux 8.6(64bit) + VEOS 3.0.3
コンパイラ	NEC SDK for Vector Engine
MPI	NEC MPI

ソフトウェア環境としては、ベクトルエンジン上で実行可能な C/C++/FORTRAN プログラムの開発環境である「NEC SDK for Vector Engine」が利用可能です。ベクトルエンジン上のプログラムから直接 MPI 通信を可能とする「NEC MPI」を標準 MPI としています。

1.2.3. GPGPU 計算環境

GPGPU 計算環境の GPU ノードは、「NEC LX 106Rj-4G」42 ノードで構成されます。CPU として「Intel Xeon Platinum 8368」、GPU として「NVIDIA A100」を有し、4U の筐体に 8GPU を搭載しています。8 つの GPU は、Delta Baseboard 上の NVSwitch により高速に相互接続されます。



GPUノード群
42ノード (7Rack)

GPUノード
CPU 2.4GHz/38core x2
GPU 9.7TFlops x8 + NVSwitch

・GPU ノード

項目		構成
総ノード数		42 ノード
サーバ構成	プロセッサ	Intel Xeon Platinum 8368(2.4 GHz/38 コア)　x2
	メモリ構成	512 GiB (32GiB DDR4-3200 ECC RDIMM x16)
	ハードディスク	240GB SATA SSD x1
	インタフェース	InfiniBand HDR100 x4, 1000BASE-T x1, BMC x1
	GPGPU	NVIDIA A100 (SXM4) x8

・ソフトウェア環境

項目	構成
OS	Rocky Linux 8.6 (64bit)
コンパイラ	NVIDIA HPC SDK
MPI	OpenMPI

ソフトウェア環境としては、GPGPU 向けプログラム開発環境である CUDA を含み、C/C++/Fortran プログラムを開発可能な「NVIDIA HPC SDK」をコンパイラとして利用可能です。また、同コンパイラと親和性の高い「OpenMPI」を標準 MPI としています。

1.3. 3 つのストレージ領域

1.3.1. 各環境の特徴

SQUID では、3 つのストレージ環境が利用可能です。それぞれの特徴は以下の通りです。

ストレージ領域	SSD ストレージ	HDD ストレージ	アーカイブ ストレージ
特徴	All Flash による超高速 IO 領域	高速かつ大容量のデータ格納領域	複製や暗号化機能を有する柔軟なアーカイブ領域
容量	(申請により追加購入)	home：10GiB work：5 TiB+追加購入	別途利用申請
ファイルシステム	DDN ExaScaler (Lustre)	DDN ExaScaler (Lustre)	Cloudian HyperStore
総容量	1.2 PB	20PB	1200TB(物理容量)
ディスク装置	15.36TB NVMe SSD	16TB 7,200rpm NL-SAS	960GB SSD(メタデータ) 10TB SAS

ハードウェア	DDN ES400NVX x5	DDN ES7990X x10	Cloudian HyperStore Appliance 1610 x10

※ 計算環境から直接利用可能なのは、SSD ならびに HDD です。

1.3.2. 様々なデータアクセス方法

SQUID では、データ利活用の向上を目的として、様々なデータアクセス方法を提供しています。各アクセス方法の参照先を下表に示します。

アクセス方法	プロトコル	用途	参照先
高速並列アクセス	Lustre	システム内での高速かつ並列アクセス	3.1.1 ファイルシステムの利用方法
CLI アクセス	SCP/SFTP	CLI によるシステム外部とのデータ転送方法	2.3 ファイル転送方法
ブラウザアクセス	HTTPS	WEB ブラウザによるシステム外部とのデータ転送方法	7.1Web ブラウザでのファイル転送
S3 アクセス	S3	S3 API による外部アプリケーションからのデータ転送方法	7.2 S3 API でのファイル転送
システム間アクセス	NFS	周辺システム(OCTOPUS)とのデータ交換方法	7.3 OCTOPUS からのファイル転送
CIFS アクセス	CIFS	大阪大学総合情報通信システム(ODINS)でのデータ連携	7.4 CIFS でのファイル転送

1.4. 3 つのフロントエンド

1.4.1. 各フロントエンドの特徴

SQUID では、3 種のフロントエンドが利用可能です。それぞれの特徴は以下の通りです。

環境	HPC フロントエンド	HPDA フロントエンド	セキュアフロントエンド
特徴	HPC アプリケーション開発向け環境。 可視化処理環境、バッチジョブ投入環境	HPDA 向け Notebook 環境 バッチジョブ投入環境	仮想マシンによる専用フロントエンド環境 バッチジョブ投入環境
ノード数	4 ノード	4 ノード	(個別申請により増減)
アプリケーション	NICE DCV Server	Jupyter	
その他	NVIDIA Quadro RTX600 x1	512GiB メモリ (HPC 用の 2 倍)	別途利用申請

1.4.2. HPC フロントエンド

HPC フロントエンドは、「NEC LX 112Rj-2」4 ノードで構成されます。CPU として「Intel Xeon Platinum」を有し、リモート可視化処理用のグラフィックアクセラレータ、可視化アプリケーション等が利用可能です。

・HPC フロントエンド

項目		構成
総ノード数		4 ノード
サーバ構成	プロセッサ	Intel Xeon Platinum 8368(2.4 GHz/38 コア)　x2
	メモリ構成	256 GiB (16GiB DDR4-3200 RDIMM x16)
	ハードディスク	960GB SSD x2
	インタフェース	InfiniBand HDR x1、 10GBASE-T x2、1000BASE-T x1、IPMI2.0 x1
	GPU	NVIDIA Quadro RTX6000 x1

・ソフトウェア環境

項目	構成
OS	RHEL 8.6 (64bit)
ジョブ投入環境	NEC NQSV
リモート可視化アプリケーション	NICE DCV Server

ソフトウェア環境は、ジョブ投入環境として、「NEC NQSV」が利用可能です。また、HPC フロントエンド上のデスクトップ画面をリモートに表示させて、HPC フロントエンド上の GPU を使って可視化処理が可能な「NICE DCV Server」の利用が可能です。

ジョブ投入環境の利用方法は「5 プログラム実行方法」を、リモート可視化アプリケーションの利用方法は「3.2 HPC 用フロントエンドの利用方法」をご参照ください。

1.4.3. HPDA フロントエンド

HPDA フロントエンドは、「NEC LX 112Rj-2」4 ノードで構成されます。CPU として「Intel Xeon Platinum」を有します。エンドユーザによって個別にカスタマイズ可能な Notebook 環境が提供されており、データ分析作業を効率的に行えます。

・HPDA フロントエンド

項目		構成
総ノード数		4 ノード
サーバ構成	プロセッサ	Intel Xeon Platinum 8368(2.4 GHz/38 コア)　x2
	メモリ構成	512 GiB (32GiB DDR4-3200 RDIMM x16)
	ハードディスク	1TB SAS HDD x2
	インタフェース	InfiniBand HDR x1、 10GBASE-T x2、1000BASE-T x1、IPMI2.0 x1

・ソフトウェア環境

項目	構成
OS	RHEL 8.6 (64bit)
ジョブ投入環境	NEC NQSV
対話型プログラム開発環境	Jupyter Notebook

ソフトウェア環境は、ジョブ投入環境として、「NEC NQSV」が利用可能です。また、WEB ブラウザ上で対話型のプログラム開発・実行環境を提供する「Jupyter Notebook」の利用できます。

ジョブ投入環境の利用方法は「5 プログラム実行方法」を、対話型プログラム開発環境の利用方法は「3.3 HPDA 用フロントエンドの利用方法」をご参照ください。

1.5. システム利用環境

1.5.1. ログイン方法

本システムでは、セキュリティ強化の一環として、ssh ログイン時の 2 段階認証を採用しています。ログインの際には、2 段階認証に必要となるアプリを用意し、ご自身の端末に事前インストールしていただく必要があります。ログインの具体的な手順については、「2 フロントエンドへのログイン」をご参照ください。

1.5.2. プログラム開発

本システムでは、C/C++/FORTRAN 言語をはじめ、各種プログラム開発を行うためのコンパイラ、ライブラリを利用できます。また、各種ビルド済みのアプリケーションの利用も可能です。

これらのアプリケーション利用に伴う環境変数設定を「Environment Module」で管理しています。module コマンドを使用することで、アプリケーションの利用に必要な環境変数を統一的に設定することができます。

また、基本的な利用の上で必要となる環境変数設定をまとめた、ベース環境を用意しています。最初にベース環境をロードすることで、簡単に必要最小限の利用環境を整備することが可能です。

本システムで用意しているベース環境は以下の通りです。

種別	ベース環境 モジュール名	内容
コンパイラ＋ MPI 環境 ＋ ライブラリ	BaseCPU/2025	汎用 CPU 計算環境向けプログラム開発の推奨環境
	BaseVEC/2025	ベクトル計算環境向けプログラム開発の推奨環境
	BaseGPU/2025	GPGPU 計算環境向けプログラム開発の推奨環境
	BaseGCC/2025	GCC を利用する際のプログラム開発環境
言語環境＋ モジュール	BasePy/2025	Python 言語向けのプログラム開発環境
	BaseR/2025	R 言語向けのプログラム開発環境
	BaseJDK/2025	JAVA 言語向けのプログラム開発環境
	BaseJulia/2025	Julia 言語向けのプログラム開発環境
アプリケーション環境	BaseApp/2025	ISV 及び OSS アプリケーション利用者向けのベース環境

「1.2 3 つの計算環境」の推奨の開発環境も、ベース環境を適用することで利用できます。このベース環境をご利用いただくことで、それぞれのハードウェアに、より適したプログラム開発を簡易に行えます。

環境設定の具体的な利用方法は、「3.1.3 環境設定」と「4 プログラム開発」をご参照ください。

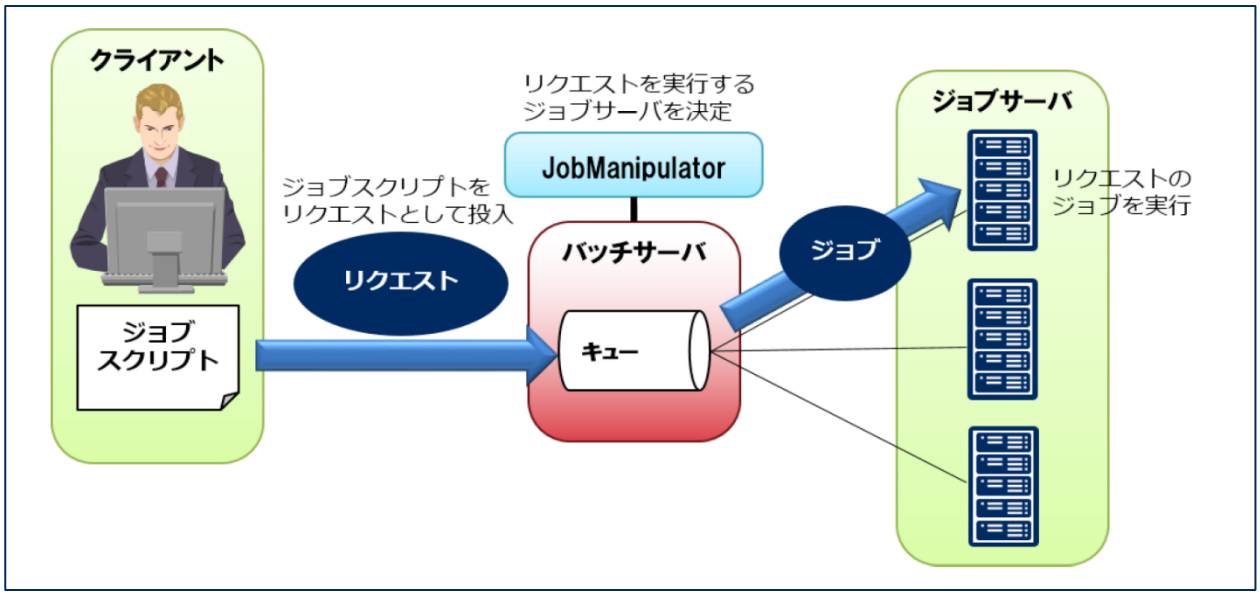
- Environment Modules 公式ページ
  http://modules.sourceforge.net/

1.5.3. プログラム実行

プログラム実行の際は、計算環境を多数のユーザで共有利用することになります。ユーザ間でプログラム実行の干渉を起こさないようにするため、実行の順序制御、ノード制御を行うために、ジョブ管理システム「NEC NQSV」が備わっています。

本システムでは、「NEC NQSV」により、計算リソースのバッチ利用および会話的利用が可能です。フロントエンドからジョブ管理システムにジョブ実行を要求（ジョブを投入）します。投入されたジョブ要求は、他のジョブ要求の優先度や要求資源量、SQUID の利用状況などがジョブ管理システムによって加味・判断され、実行順序が決定されます。

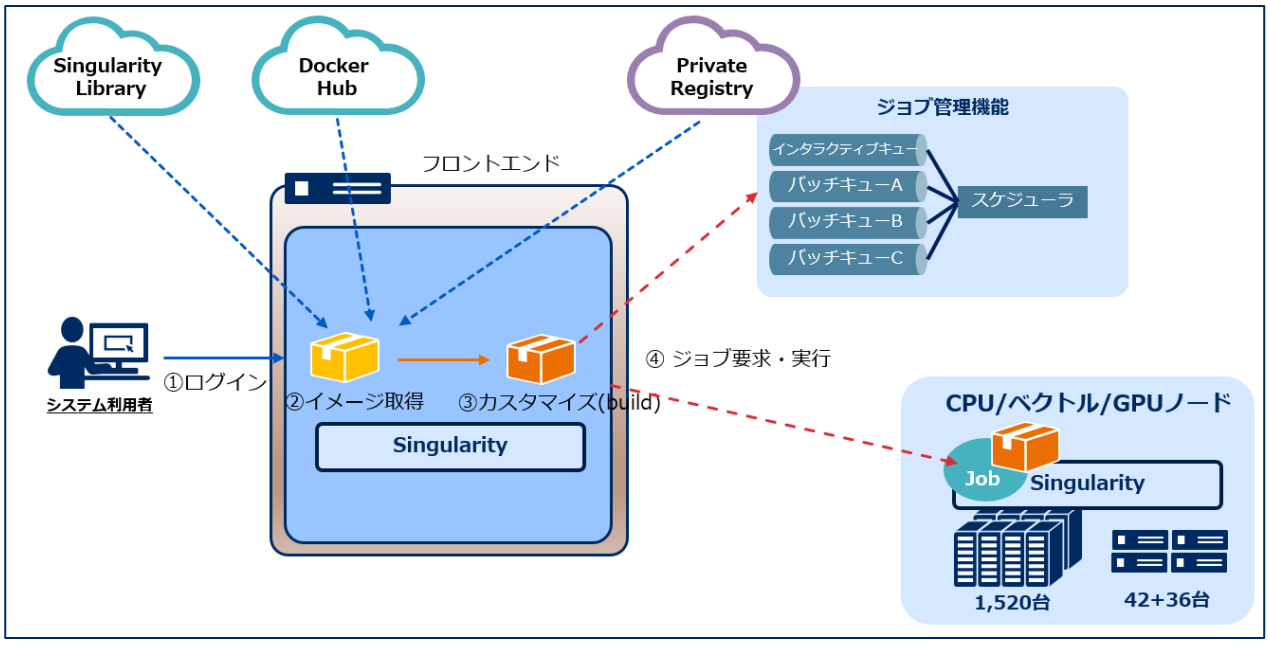
SQUID では、多人数のユーザで共有利用されることが前提となっているため、このようなバッチ処理方式を主なプログラム実行方針として採用しています。



ジョブ管理システムの具体的な利用方法は、「5 プログラム実行方法」をご参照ください。

1.5.4. コンテナ利用

本システムでは、Singularity を用いたコンテナの利用が可能です。インターネット上やシステム内で公開されているコンテナイメージを、ホームディレクトリへ展開し、カスタマイズした上で、ジョブとして実行することが可能です。



構成済みのソフトウェア群をコンテナとして利用することで、事前準備の手間の削減、プログラム実行の再現性向上、複雑な予備知識なしでも利用可能といった効果があります。

コンテナの具体的な利用方法は、「4.7 コンテナの利用方法」をご参照ください。また、コンテナを用いたジョブの実行例は、「5.6 コンテナの実行方法」も合わせてご参照ください。

以下に各手順の概要を記載します。

(1) イメージ取得

Singularity のコンテナは、慣例的には .sif という拡張子で表されるイメージファイルを用意して利用します。イメージファイルは、インターネット上やシステム内で公開されているものをダウンロードしたり、ユーザ自身で作成したファイルをシステム内に転送したりといったことが可能です。本章の「4.7.1 コンテナイメージの準備」では、以下の手順について説明しています。

- ローカルのワークステーションからシステム内に転送する
- SQUID ローカルレジストリからイメージを取得する
- Singularity Library からイメージを取得する
- Docker Hub からイメージを取得する
- 参考：NVIDIA GPU CLOUD(NGC)からイメージを取得する

各コミュニティで公開されているコンテナが、本システムで必ずしも動作するとは限りませんが、実行環境を整える上で有益であるため、各種コミュニティを利用したイメージ取得方法を本書内で説明しています。

(2) カスタマイズ(build)

実行したい内容に合わせて、必要に応じてイメージファイルをカスタマイズします。変更内容をイメージファイルに反映する作業を build といいます。本章の「4.7.2 コンテナイメージのカスタマイズとビルド」では、システム内で行えるカスタマイズ手段として、以下の手順を説明しています。

- sandbox 経由でカスタマイズする方法
- def ファイルにカスタマイズ内容を記載する方法

sandbox 経由でカスタマイズする方法は、通常の Unix シェルの感覚でカスタマイズ可能ですが、利用者は root 権限がないため制約を受けるケースがあります。可能であれば、root 権限をもつご自身のローカルのワークステーション上でカスタマイズし、システム内にイメージファイルを転送する方が、より制約を受けることなくカスタマイズ可能です。

(3) ジョブ要求・実行

システム内でのプログラム実行は、ジョブ管理システムへのジョブ要求による実行が基本となります。「5.6 コンテナの実行方法」では、コンテナを利用したジョブの実行手順について説明しています。

Singularity に関する詳細な内容は、man コマンドを参照いただくか、ドキュメントがインターネット上でも公開されていますので、ご参照ください。

- Singularity 公式ページ
  https://sylabs.io/singularity/

- Singularity 3.7 User Guide
  https://sylabs.io/guides/3.7/user-guide/

2. フロントエンドへのログイン

2.1. SSH でのログイン方法

SQUID にアクセスするためには、大規模計算機システムが接続されたスーパーコンピュータネットワークである CMC-Scinet にアクセスする必要があります。本システムでは SQUID へのアクセスのために、HPC フロントエンド、HPDA フロントエンドを設けています。

ログインに必要な接続情報、接続先のホスト名は以下の通りです。

項	サーバ	ホスト名
1	HPC フロントエンド	squidhpc.hpc.cmc.osaka-u.ac.jp
2	HPDA フロントエンド	squidhpda.hpc.cmc.osaka-u.ac.jp

上記のホスト名に対して、以下の接続方法でアクセスしてください。

接続方法	SSH
認証方法	2 段階認証
OS 日本語コード	ja_JP.UTF-8

2.1.1. 2 段階認証アプリのインストール

フロントエンドに SSH ログインするためには、Google Authenticator 等を使った 2 段階認証を通る必要があります。2 段階認証に必要となるアプリケーションを用意し、**ご自身の端末、スマートフォンにインストール**してください。

2 段階認証アプリケーションとしては、以下が使用できることを確認済みです。

OS	アプリケーション	備考
Android	Google Authenticator	Google Play Store
iOS	Google Authenticator	Apple App Store
Windows	WinAuth	https://winauth.github.io/winauth/download.html
macOS	Step Two	Apple App Store

以降の手順では、iOS 版 Google Authenticator を例として説明します。

2.1.2. 初回のログイン

SSH コマンドまたはターミナルソフトウェアを用いて、フロントエンドに SSH アクセスします。

(1) 初回 SSH アクセス

- **UNIX 系 OS(Linux、macOS)からのログイン**

ssh コマンドを使います。

HPC フロンエンド(squidhpc.hpc.cmc.osaka-u.ac.jp)にログインする例は以下の通りです。

```
$ ssh (-l ユーザ名) squidhpc.hpc.cmc.osaka-u.ac.jp
The authenticity of host 'squidhpc.hpc.cmc.osaka-u.ac.jp (133.1.66.X)' can't be established.
RSA key fingerprint is 32:fd:73:4e:7f:aa:5d:3c:2e:ab:37:83:d6:55:98:e2.
Are you sure you want to continue connecting (yes/no)? yes
                                    (最初のアクセス時のみ問い合わせがあります)
(ユーザ名)@squidhpc.hpc.cmc.osaka-u.ac.jp's password: *****
                                    (利用者管理システムと同じパスワードを入力します。)
```

※(-l ユーザ名):ログインユーザ名がローカルマシンのユーザ名と異なる場合に指定

- **Windows マシンからのログイン**

フリーソフトウェア「Tera Term Pro」を用いた例を説明します。

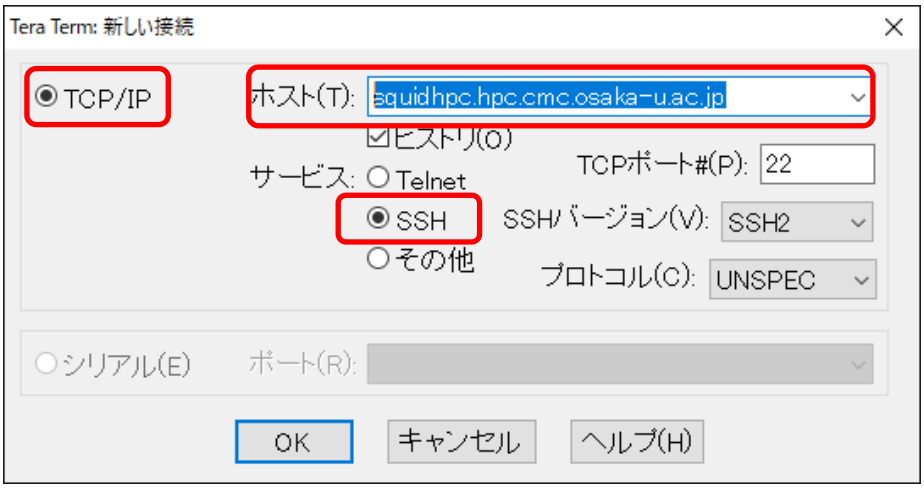
「Tera Term Pro」を起動し、「Tera Term: New connection ダイアログ」において

・TCP/IP を選択します。

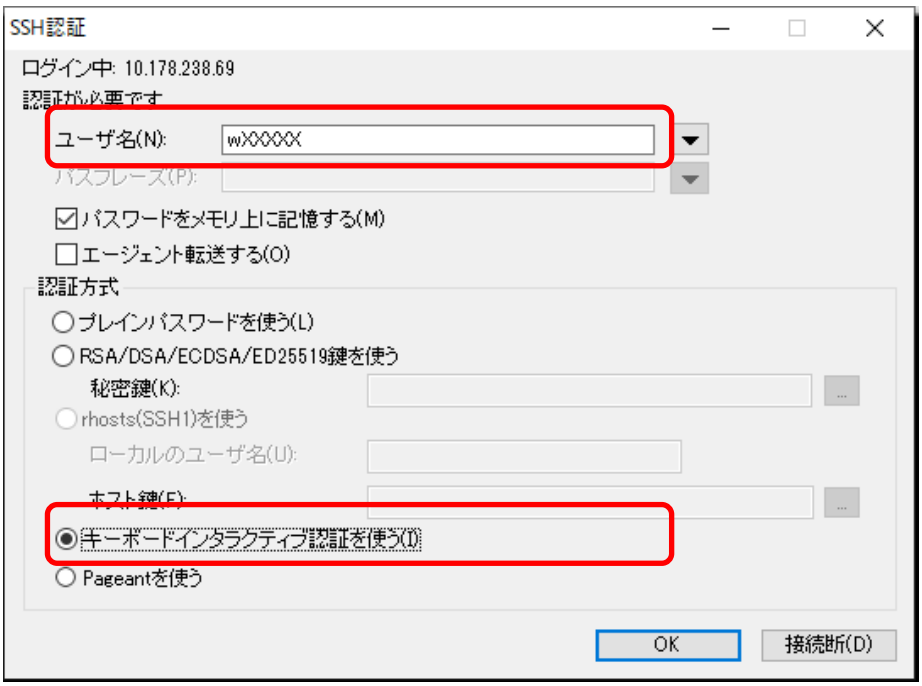
・ホスト欄にホスト名(ここでは squidhpc.hpc.cmc.osaka-u.ac.jp)を入力します。

・サービスは SSH を選択します。

・OK をクリックします。



SSH 認証画面でユーザ名を入力し、認証方式として「キーボードインタラクティブ認証を使う」を選択します。



SSH 認証チャレンジ画面で、利用者管理システムと同じパスワードを入力します。

(2) 2 段階認証の初期設定

初回ログインした際は、ターミナルに以下のように表示されます。

```
Initiallize google-authenticator
Warning: pasting the following URL info your browser exposes the OTP secret to Google:

https://www.google.com/chart?chs=200*200&chld=M|0&cht=qr&otpauth://totp/user1@squidhpc.hpc.cmc.osaka-u.ac.jp%3Fsecret%3DDXXXXXXXXCLI%26issuer%3Dsquidhpc.hpc.cmc.osaka-u.ac.jp
```

```
Your new secret key is: XXXXXXXXXX
Enter code from app (-1 to skip): -1
Code confirmation skipped
Your emergency scratch codes are:
```

画面に表示される QR コードまたは「Your new secret key is:」に続く secret key を 2 段階認証アプリケーションの登録に使用します。

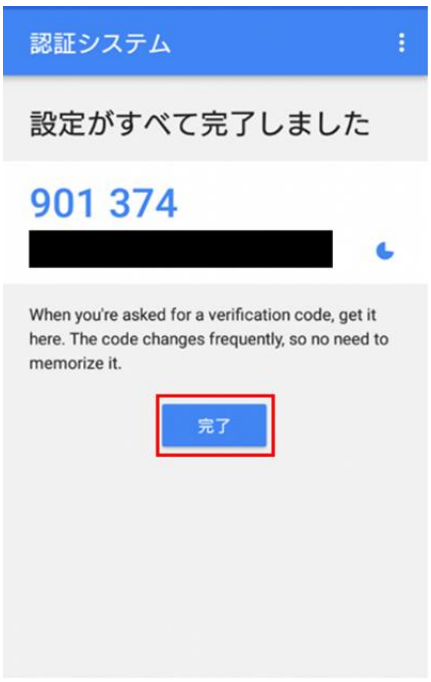
※　ターミナルソフトのウィンドウサイズによっては QR コードの表記が崩れる場合がございます。フォントサイズを調整するか、記載の URL、シークレットキーをお使いください。

(3) 2 段階認証アプリの設定

① Google Authenticator アプリを起動し「開始」ボタンをクリックします。

② 「バーコードをスキャン」または「提供されたキーを入力」を選択し(2)で表示された QR コードまたはシークレットキーを入力します。

③ 完了すると、Google 認証システムに対象のユーザが登録され、ワンタイムパスワードが発行されます。



(4) 初回ログイン終了

ターミナルに戻り、質問に対して入力します。

```
Your new secret key is: XXXXXXXXXXX
Enter code from app (-1 to skip): -1
Code confirmation skipped
Your emergency scratch codes are:
ok? Hit enter: (Enter キー)
```

「Enter code from app (-1 to skip):」に対しては**-1 を入力しスキップしてください。**

「ok? Hit enter:」が表示されます。Enter キーを押すと 1 度ログアウトします。

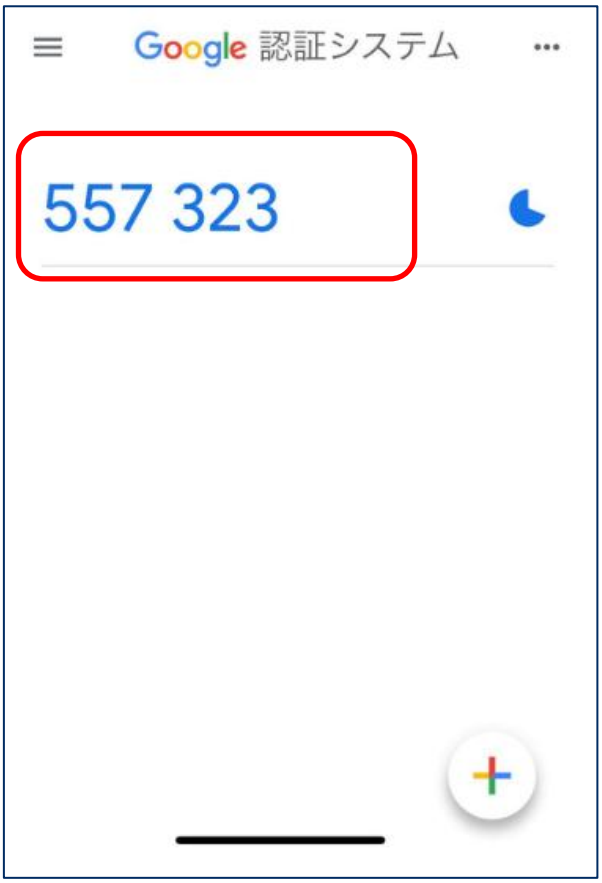
※「Your emergency scratch codes are:」では、1 度だけ使用可能なコードをシステムで指定した件数分だけ発行することが可能です。しかし、SQUID においてはセキュリティの観点から 0 件で設定して使用不可としています。

2.1.3. 2 回目以降のログイン

初回ログインにて登録したワンタイムパスワードを用いて、SSH ログインを行います。

(1) 2 段階認証アプリの起動

2 段階認証アプリを起動して、ワンタイムパスワードを確認します。



※ワンタイムパスワードは、30 秒ごとに新しくなります。ログイン時には最新のワンタイムパスワードを使用するため、ログイン完了まで、上記の画面を表示したままにしておきます。

(2) SSH アクセス

ターミナルソフトまたは SSH コマンドを利用して、SSH アクセスを行います。ログイン時に、(1)で取得したワンタイムパスワードを利用します。

- **UNIX 系 OS(Linux、macOS)からのログイン**

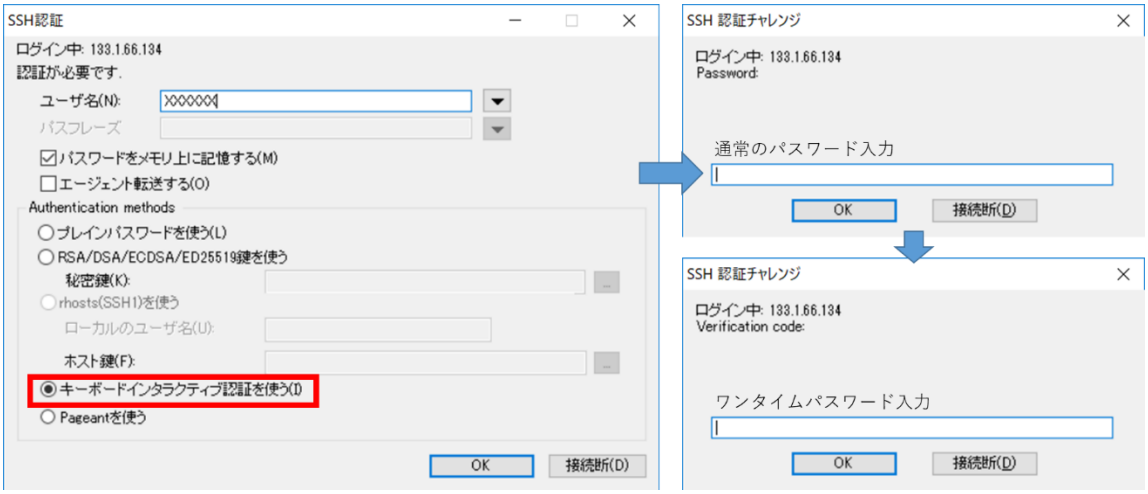
ssh コマンドを使います。フロンエンド(squidhpc.hpc.cmc.osaka-u.ac.jp)に SSH ログインする例は以下の通りです。

```
$ ssh (-l ユーザ名) squidhpc.hpc.cmc.osaka-u.ac.jp
(ユーザ名)@squidhpc.hpc.cmc.osaka-u.ac.jp's password: *****
                        (利用者管理システムと同じパスワードを入力します。)
Verification code: *****
                        (ワンタイムパスワードを入力)


※(-l ユーザ名):ログインユーザ名がローカルマシンのユーザ名と異なる場合に指定
```

➢ **Windows マシンからのログイン**

フリーソフトウェア「Tera Term Pro」を用いた例を説明します。SSH 認証チャレンジの部分で、ワンタイムパスワードを入力しログインします。



2.1.4. 2 段階認証できなくなった場合の復旧方法

管理者操作が必要となりますので、システム管理者へご連絡ください。

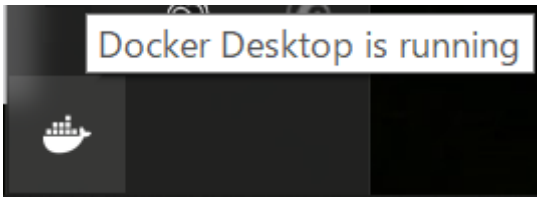
2.2. GSI-SSH でのログイン方法

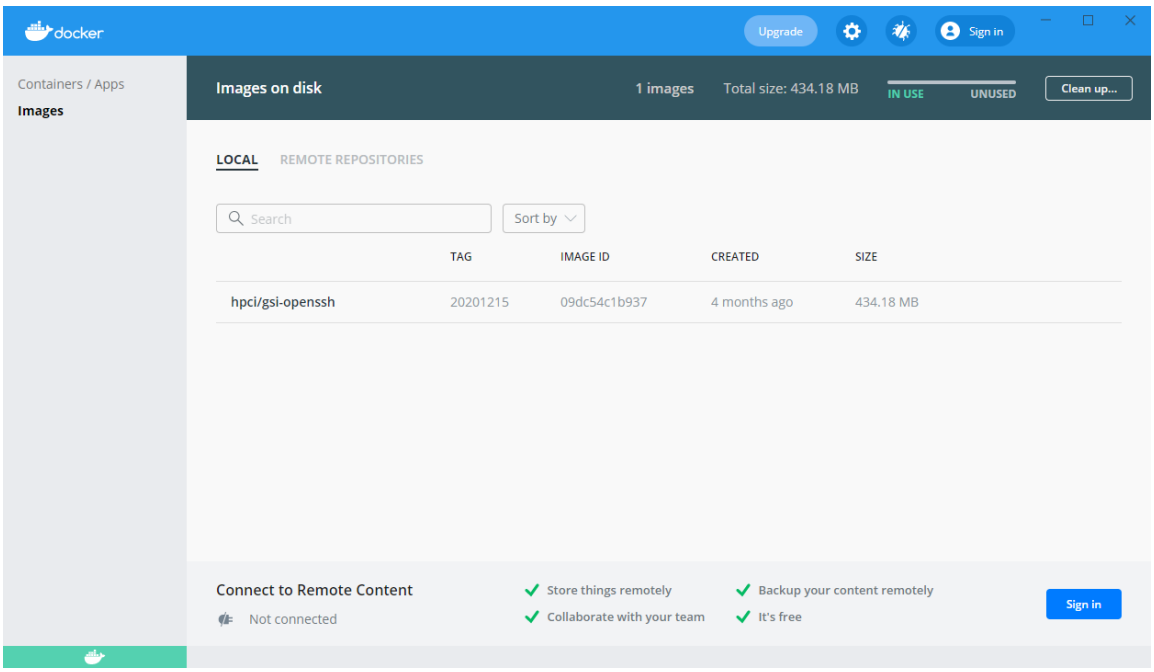
HPCI 利用者の場合、GSI 認証によるログインが可能です。ここでは Windows10 からのログイン方法をご紹介します。

GSI 認証でログインする場合は、事前に、HPCI 運用事務局が公開しているユーザマニュアル『HPCI ログインマニュアル(HPCI-CA01-001-21)』の「2.1. 電子証明書の発行手順」をご参照のうえ、電子証明書と代理証明書を発行しておいてください。また、同マニュアルの「1.3.2. Docker Container image for GSI-OpenSSH」に従い、「Docker Container image for GSI-OpenSSH」をインストールしてください。

(1) Docker Desktop の起動

「Docker Desktop for Windows」が起動していることを確認します。





(2) Windows PowerShell の起動

docker コマンドを実行するため、「Windows PowerShell」を起動します。「Windows PowerShell」は一般ユーザ権限で起動してください。

(3) Docker イメージのロード (導入時のみ)

「Windows PowerShell」で以下のコマンドを実行し、Docker イメージをロードします。

```
PS C:¥Users¥username> docker load -i gsi-openssh-20201215.tar.bz2
c33ea345ab20: Loading layer [==================================>]  242.3MB/242.3MB
039c616acc15: Loading layer [==================================>]  31.23kB/31.23kB
7bf8d15f20d6: Loading layer [==================================>]  4.096kB/4.096kB
dd7e690f7986: Loading layer [==================================>]  737.3kB/737.3kB
Loaded image: hpci/gsi-openssh:20201215
PS C:¥Users¥username>
PS C:¥Users¥username> docker images hpci/gsi-openssh
REPOSITORY          TAG         IMAGE ID       CREATED         SIZE
hpci/gsi-openssh    20201215    09dc54c1b937   3 months ago    434MB
```

(4) Docker コンテナの起動

「Windows PowerShell」で以下のコマンドを実行し、ロードした Docker コンテナを起動します。

```
PS C:¥Users¥username> docker run -d --rm --name gsi-openssh -v
c:¥Users¥username¥Documents:/home/hpciuser/work hpci/gsi-openssh:20201215
61506efcd6a00f422aeacf04fb4819307f591b173b79b06fdaa5b64c1ac1b827
PS C:¥Users¥username>
```

(5) GSI-OpenSSH bash の開始

以下のコマンドを実行し、GSI-OpenSSH を/bin/bash で実行します。

```
PS C:¥Users¥username> docker exec -i -t gsi-openssh /bin/bash
[hpciuser@a4dc58b0dee1 ~]$
```

(6) 代理証明書のダウンロード

以下のコマンドを実行し、代理証明書をダウンロードします。

```
[hpciuser@a4dc58b0dee1 ~]$ myproxy-logon -s portal.hpci.nii.ac.jp -l hpciXXXXXX
                                                                    課題 ID
```

以下のコマンドを実行し、取得した代理証明書の情報を確認します。

```
[hpciuser@a4dc58b0dee1 ~]$ grid-proxy-info
subject  : /C=JP/O=NII/OU=HPCI/CN=username[hpciXXXXXX]/CN=XXX/CN=XXX/CN=XXX/CN=XXX
issuer   : /C=JP/O=NII/OU=HPCI/CN=username [hpciXXXXXX]/CN=XXX/CN=XXX/CN=XXX
identity : /C=JP/O=NII/OU=HPCI/CN=username [hpciXXXXXX]
type     : RFC 3820 compliant impersonation proxy
strength : 2048 bits
path     : /tmp/x509up_u2000
timeleft : 11:59:39
[hpciuser@a4dc58b0dee1 ~]$
```

(7) ログインサーバへのログイン

ログインサーバを指定し、GSI 認証でログインします。

・HPC ログインサーバにログインする場合

```
[hpciuser@a4dc58b0dee1 ~]$ gsissh -p 2222 squidhpc.hpc.cmc.osaka-u.ac.jp
Last login: Thu Dec 17 21:21:52 2020 from XX.XX.XX.XX
-bash-4.2$ pwd
/sqfs/home/username
```

・HPDA ログインサーバにログインする場合

```
[hpciuser@a4dc58b0dee1 ~]$ gsissh -p 2222 squidhpda.hpc.cmc.osaka-u.ac.jp
Last login: Thu Dec 17 21:21:52 2020 from XX.XX.XX.XX
-bash-4.2$ pwd
/sqfs/home/username
```

(8) 終了処理

以下のコマンドを実行し、ログインサーバと bash からログアウトします。

```
-bash-4.2$ exit
[hpciuser@a4dc58b0dee1 ~]$
PS C:¥Users¥username>
```

以下のコマンドを実行し、Docker コンテナを停止します。

```
PS C:¥Users¥username> docker stop gsi-openssh
gsi-openssh
PS C:¥Users¥username>
```

2.3. ファイル転送方法

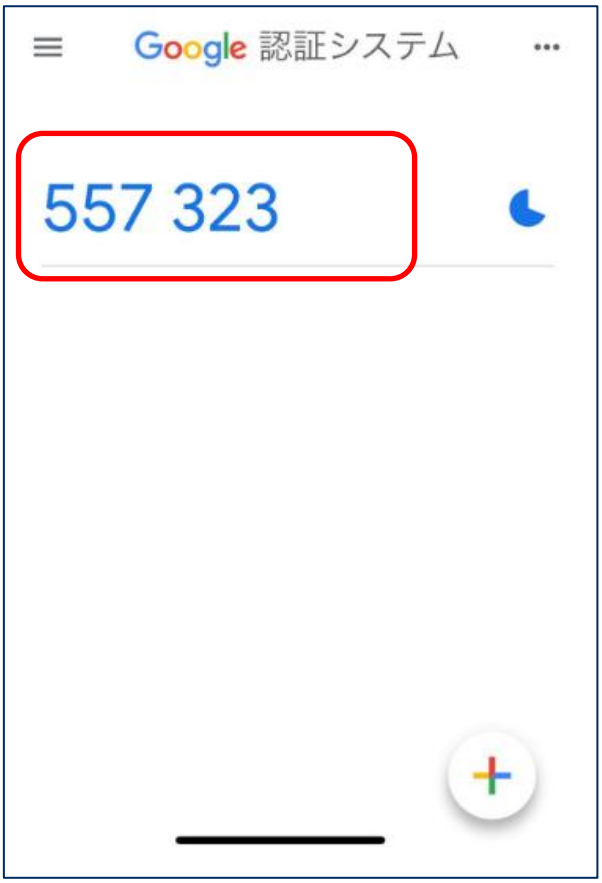
2.3.1. UNIX 系 OS（Linux, macOS）でのファイル転送

ここでは、サイバーメディアセンター外部の研究室などの UNIX 系 OS（Linux, macOS）端末から、FTP サーバへ接続する例を紹介します。端末側に予め SFTP や SCP がインストールされていることを前提としています。SFTP や SCP がインストールされていない場合は、ご自分でインストールしていただくか、端末の管理者にご相談ください。

※ファイル転送のためには、事前に 2 段階認証の初期設定が必要となります。詳細は「2.1.2 初回のログイン」の項をご参照ください。

(1) 2 段階認証アプリの起動

2 段階認証アプリを起動して、ワンタイムパスワードを確認します。



※ワンタイムパスワードは、30 秒ごとに新しくなります。ログイン時には最新のワンタイムパスワードを使用するため、ログイン完了まで、上記の画面を表示したままにしておきます。

➢ **接続手順(SFTP の場合)**

sftp コマンドを使って、FTP サーバに接続します。次の例は、ユーザ名 a61234 で接続する例です。

(1) SFTP コマンド実行

次のコマンドを入力します。

```
$ sftp a61234@squidhpc.hpc.cmc.osaka-u.ac.jp
$ sftp a61234@squidhpda.hpc.cmc.osaka-u.ac.jp
```

(2) パスワード入力

初めて FTP サーバに接続する場合は次のようなメッセージが表示されますので、yes を入力します。

```
The authenticity of host ' squidhpc.hpc.cmc.osaka-u.ac.jp' can't be established but keys of different type are already known for this host.
RSA key fingerprint is 19:14:7f:28:54:39:16:9a:99:d0:db:93:d6:ff:f3:13.
Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'XXXX.hpc.cmc.osaka-u.ac.jp,133.1.65.XX' (RSA) to the list of known hosts.
```

パスワードを聞かれますので、利用者管理システムと同じパスワードを入力すれば FTP サーバへの接続が完了します。

```
a61234@squidhpc.hpc.cmc.osaka-u.ac.jp's password:    XXXXXXXX
a61234@squidhpda.hpc.cmc.osaka-u.ac.jp's password:    XXXXXXXX
(入力したパスワードは表示されません)
```

(3) ワンタイムパスワード入力

Verification code を聞かれますので、2 段階認証アプリから取得したワンタイムパスワードを入力します。

```
Verification code: *****      (ワンタイムパスワードを入力)
```

(4) FTP 通信開始

通常の FTP と同様に、get、 put コマンドなどによりファイル転送を行います。

次の例は、sample.f ファイルを SQUID のホームに転送する例です。

```
sftp> put sample.f
Uploading sample.f to /sqfs/home/a61234/sample.f
```

- **接続手順(SCP の場合)**

scp コマンドを使ってファイル転送をする場合は、予め転送するファイル名や転送先のディレクトリ名がわかっている必要があります。

(1) SCP コマンド実行

次のコマンドを入力します。

・ ローカル端末のカレントディレクトリ上のファイル sample.f を、SQUID のホームディレクトリに転送する場合

```
$ scp sample.f a61234@squidhpc.hpc.cmc.osaka-u.ac.jp:
$ scp sample.f a61234@squidhpda.hpc.cmc.osaka-u.ac.jp:
```

・ SQUID のホームディレクトリ下の abc ディレクトリの中のファイル sample.c を、ローカル端末のカレントディレクトリ上に転送する場合

```
% scp a61234@squidhpc.hpc.cmc.osaka-u.ac.jp:abc/sample.c sample2.c
% scp a61234@squidhpda.hpc.cmc.osaka-u.ac.jp:abc/sample.c sample2.c
```

※ scp では、複数のファイルを一度に転送したり、ディレクトリごと再帰的にファイルを転送したりすることもできます。詳しくは、「man scp」コマンドで scp のマニュアルをご参照ください。

(2) パスワード入力

初めて SCP サーバに接続する場合は次のようなメッセージが表示されますので、yes を入力します。

```
The authenticity of host 'squidhpc.hpc.cmc.osaka-u.ac.jp' can't be established but keys of different type are already known for this host.
RSA key fingerprint is 19:14:7f:28:54:39:16:9a:99:d0:db:93:d6:ff:f3:13.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'XXXX.hpc.cmc.osaka-u.ac.jp,133.1.65.XX' (RSA) to the list of known hosts.
```

パスワードを聞かれますので、利用者管理システムと同じパスワードを入力します。

```
a61234@squidhpc.hpc.cmc.osaka-u.ac.jp's password:     XXXXXXXX
a61234@squidhpda.hpc.cmc.osaka-u.ac.jp's password:     XXXXXXXX
(入力したパスワードは表示されません)
```

(3) ワンタイムパスワード入力

Verification code を聞かれますので、2 段階認証アプリから取得したワンタイムパスワードを入力します。

```
Verification code: *****      (ワンタイムパスワードを入力)
```

(4) ファイル転送

ファイル転送が行われ、転送状況、転送ファイルサイズ、転送時間が表示されます。

```
sample.f 100% |*****************************| 1326 00:01
```

2.3.2. Windows でのファイル転送

SSH（ Secure Shell ） に対応したファイル転送ソフトウェアをパソコンにインストールすれば、ファイル転送サーバ（以下、FTP サーバ）への接続が可能です。該当するソフトは多数ありますが、ここではフリーソフトの「WinSCP」を用いてファイル転送をする例を紹介します。

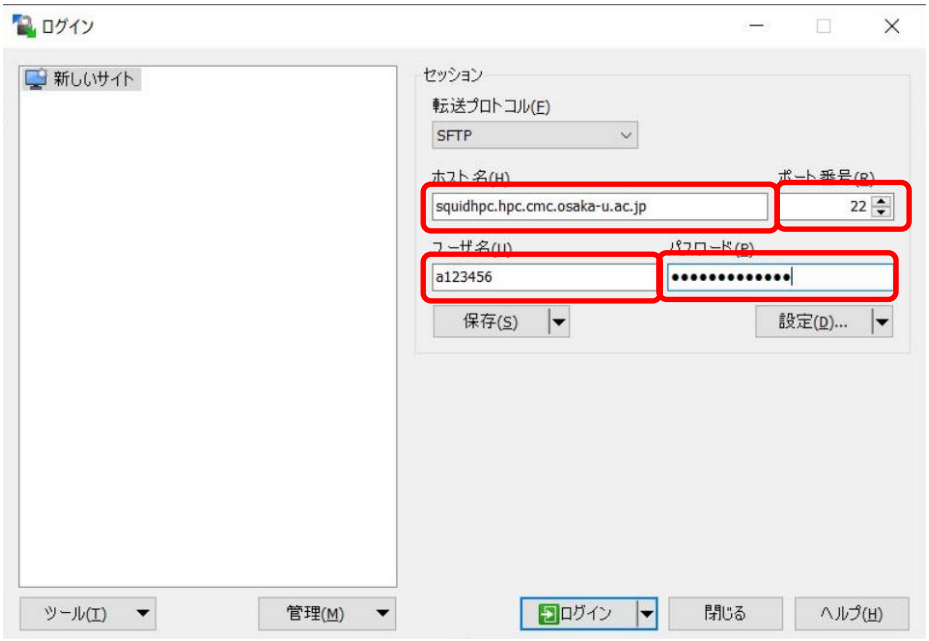
(1) WinSCP の入手

WinSCP 公式サイトなどからダウンロードし、手順に従ってインストールしてください。

(2) WinSCP の起動

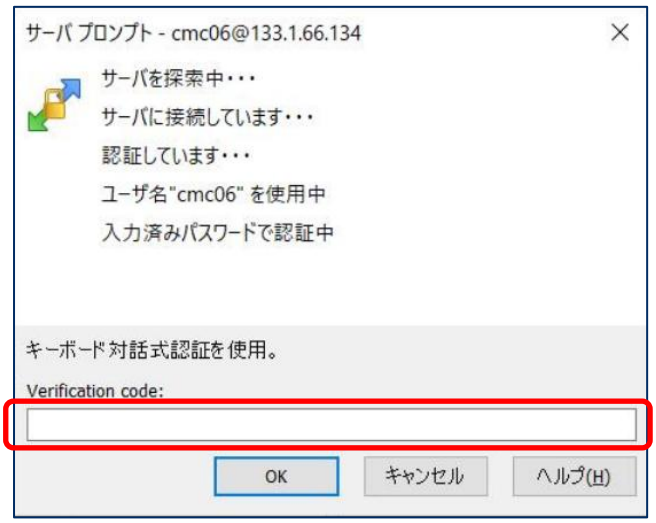
WinSCP を起動し、FTP サーバへ接続します。以下の項目を設定後、[ログイン]ボタンをクリックします。

```
ホスト名:  squidhpc.hpc.cmc.osaka-u.ac.jp または squidhpda.hpc.cmc.osaka-u.ac.jp
ポート番号: 22
ユーザ名:    利用者番号
パスワード: 利用者管理パスワード
```



(3) ワンタイムパスワード入力

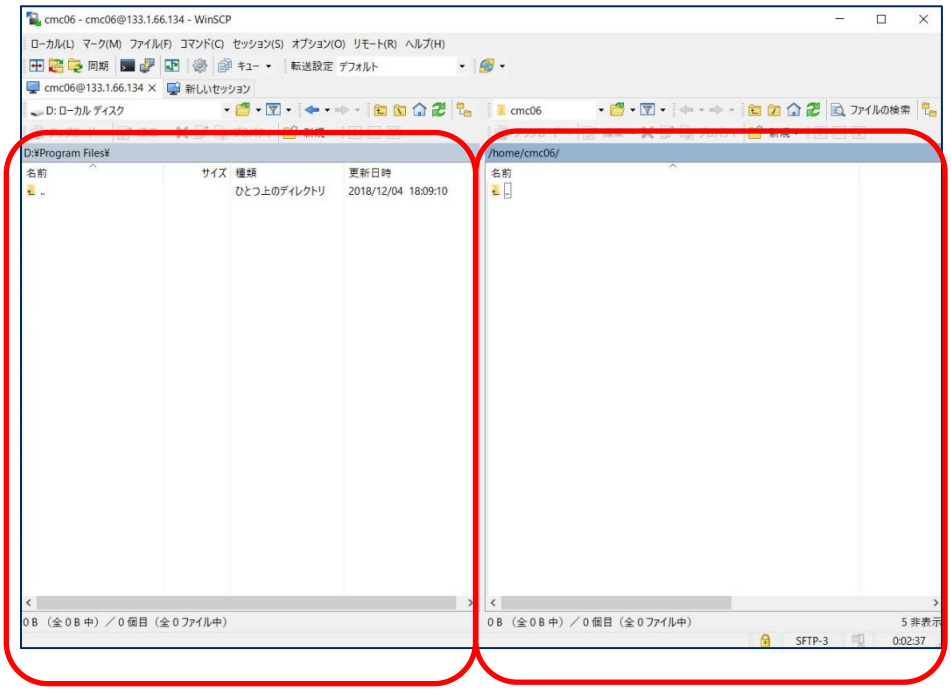
Verification code を聞かれますので、2 段階認証アプリから取得したワンタイムパスワードを入力します。



※プログラムソースや NQS スクリプトなどのテキストファイルを転送する際は、「設定」→「環境設定」の項目にて、転送モードを「テキスト」に設定してください。 なお、設定変更後は、セッション情報を「保存」しておくと便利です。

(4) ファイル転送開始

ログインに成功すると、ローカル(Windows)のフォルダと、リモート(SQUID)のフォルダが左右に表示されます。



ローカル(Windows)側　　**リモート(SQUID)側**

転送対象ファイルを左右に、ドラッグアンドドロップすることで、ファイル転送が可能になります。

3. フロントエンド環境

3.1. 共通事項

3.1.1. ファイルシステムの利用方法

フロントエンド環境からは、ファイルシステムとして SSD 領域および HDD 領域が直接利用できます。割り当てられているディスク領域およびそのクォータ制限は以下の通りです。

ストレージ	ファイルシステム	領域名	パス	クォータ		備考
				サイズ	ファイル	
HDD	EXAScaler (Lustre)	ホーム領域	/sqfs/home/(利用者番号)	10GiB	---	ブラウザアクセス領域を含む
		拡張領域	/sqfs/work/(グループ名)/(利用者番号)	5TiB (+)	---	追加購入可
			/sqfs/s3/(UUID)			S3 アクセス用
SSD	EXAScaler (Lustre)	高速領域	/sqfs/ssd/(グループ名)/(利用者番号)	0B (+)	---	追加購入可

- **ホーム領域**

ユーザ登録時に与えられる基本領域です。初期容量として、10GiB が割り当てられます。

ファイルパス(home)は以下になります。

```
/sqfs/home/(利用者番号)
```

例: 利用者番号"user001"の場合　/sqfs/home/user001

- **拡張領域**

高速かつ大容量の I/O が可能な領域です。以下の特徴があります。

- 初期容量は 5TiB の割り当てがあります。追加購入いただくことによりグループ毎に容量を増やすことができます。
- SQUID 外部から S3 API でアクセスするための領域が、別パスで用意されます。

ファイルパス(work)は以下になります。

```
/sqfs/work/(グループ名)/(利用者番号)
```

例: 利用者番号"user001"、グループ名"G012345"の場合　/sqfs/work/G012345/user001

ファイルパス(s3)は以下になります。

```
/sqfs/s3/(UUID)
```

※ UUID は、S3 API アクセスに利用するアクセスキー毎に採番される識別情報で、アクセスキー生

成時に確認が可能です。アクセスキーについては「7.2.1 アクセスキーの発行」の項をご参照ください。

- **高速領域**

SSD のストレージで、さらに高速な I/O が可能な領域です。以下の特徴があります。

- SSD ストレージによる超高速ファイルシステム
- 初期容量は割り当てがありません。追加購入により所属グループ毎に容量を増やすこと可能です。

ファイルパス(ssd)は以下になります。

/sqfs/ssd/(グループ名)/(利用者番号)

例: 利用者番号"user001"、グループ名"G012345"の場合　/sqfs/ssd/G012345/user001

3.1.2. 利用状況の確認方法

フロントエンド環境から計算機・ファイルシステムの利用状況を確認するためのコマンドとして usage_view を用意しております。

usage_view コマンドを実行すると以下のように表示されます。

```
$ usage_view
---------------+ Group: G012345 +---------------

[Group summary]

          SQUID points   HDD(GiB)   SSD(GiB)
------- -------------- ---------- ----------
usage              0.0       87.9        0.0
limit          11000.0    46080.0    20480.0
remain         11000.0    45992.1    20480.0
rate(%)            0.0        0.2        0.0


[Detail]

          SQUID points              Home(GiB)   HDD(GiB)   SSD(GiB)
------- -------------- -------------------- ---------- ----------
user001            0.0   0.0 /  10.0 /  10.0        0.0        0.0
user002            0.0   0.0 /  10.0 /  10.0        0.0        0.0
user003            0.0   0.0 /  10.0 /  10.0        0.0        0.0

[Node-hours]

                             Usage      Available*
------------------ -------------- --------------
CPU           node           0.00        9090.90
GPU           node           0.00        8184.52
Vector        node           0.00        8461.53
Azure  CPU    node           0.00        6547.61
       GPU    node           0.00           0.00
OCI    CPU    node           0.00        6547.61
       GPU    node           0.00           0.00
Secure CPU    node           0.00        6666.66
       GPU    node           0.00        6666.66
       Vector node           0.00        6666.66

 * Available Node-hour is estimated based on remains of SQUID, Bonus point.
```

上記で表示される項目は以下の通りです。

[Group summary]

- SQUID points
  グループが現在までに使用したポイント、申請したポイント、使用可能な残りポイント、使用率を表示しています。
- Bonus points
  グループが現在までに使用したボーナスポイント、割り当てられているボーナスポイント、使用可能な残りボーナスポイント、使用率を表示しています。（ボーナスポイントは試用制度利用時等に付与されるポイントです）
- HDD(GiB)
  グループが使用している拡張領域のディスク使用量、使用可能な総ディスク容量、使用可能な残りディスク容量、使用率を表示しています。
  ※ディスク容量はホーム領域と拡張領域の合算値が表示されます。
- SSD(GiB)
  グループが使用している高速領域のディスク使用量、使用可能な総ディスク容量、使用可能な残りディスク容量、使用率を表示しています。

[Detail]

- SQUID points
  ユーザ毎に、ユーザが現在までに使用したポイント、申請したポイント、使用可能な残りポイント、使用率を表示しています。
- Home(GiB)
  ユーザ毎に、ユーザが使用しているホーム領域のディスク使用量／使用可能な総ディスク使用量／使用可能な残りディスク容量を表示しています。
- HDD(GiB)
  ユーザ毎に、ユーザが使用している拡張領域のディスク使用量を表示しています。
  ※ディスク容量はホーム領域と拡張領域の合算値が表示されます。
- SSD(GiB)
  ユーザ毎に、ユーザが使用している高速領域のディスク使用量を表示しています。

[Node-hours]

- Usage
  グループが現在までに使用したノード時間を、ノード群別に表示します。
- Usage(Bonus)

グループがボーナスポイントで現在までに使用したノード時間を、ノード群別に表示します。

- Available
  使用可能な残り SQUID ポイントを、使用可能な残りノード時間へ変換した値を表示します。残り SQUID ポイントを全てそのノードで使用した場合のノード時間です。
- Available(Bonus)
  使用可能な残り SQUID ボーナスポイントを、使用可能な残りノード時間へ変換した値を表示します。残り SQUID ボーナスポイントを全てそのノードで使用した場合のノード時間です。

3.1.3. 環境設定

本システムではコンパイラ、ライブラリ、アプリケーションの環境変数設定を「Environment modules」で管理しています。module コマンドを使用することでアプリケーションの利用に必要な環境変数を統一的に設定することが可能です。以下に主な利用方法を示します。

コマンド	説明
module avail	利用可能な開発環境/アプリの一覧表示
module list	ロード済みのモジュールの一覧表示
module switch [file1] [file2]	モジュールの入れ替え（file1 → file2）
module load [file]	モジュールのロード
module unload [file]	モジュールのアンロード
module purge	ロード済みの全モジュールのアンロード
module show [file]	モジュールの詳細表示

本システムでは、基本的な利用の上で必要となる環境変数設定をまとめた、ベース環境を用意しています。最初にベース環境をロードすることで、簡単に必要最小限の利用環境を準備できます。

本システムで用意しているベース環境は以下の通りです。

種別	モジュール名	内容
コンパイラ＋ MPI 環境 ＋ ライブラリ	BaseCPU/2025	汎用 CPU 計算環境向けプログラム開発の推奨環境
	BaseVEC/2025	ベクトル計算環境向けプログラム開発の推奨環境
	BaseGPU/2025	GPGPU 計算環境向けプログラム開発の推奨環境
	BaseGCC/2025	GCC を利用する際のプログラム開発環境
言語環境＋ モジュール	BasePy/2025	Python 言語向けのプログラム開発環境
	BaseR/2025	R 言語向けのプログラム開発環境
	BaseJDK/2025	JAVA 言語向けのプログラム開発環境
	BaseJulia/2025	Julia 言語向けのプログラム開発環境

アプリケーション環境	BaseApp/2025	ISV 及び OSS アプリケーション利用者向けのベース環境

以下に module コマンドの実行例を記載します。

① ロード可能なモジュール一覧を表示

```
$ module avail
--------------------------- /system/apps/env/Base ----------------------------------------
BaseApp/2021                BaseGCC/2022              BaseJulia/2023
BaseApp/2022                BaseGCC/2023              BaseJulia/2024
BaseApp/2023                BaseGCC/2024              BaseJulia/2025(default)
BaseApp/2024                BaseGCC/2025(default)     BasePy/2021
BaseApp/2025(default)       BaseGPU/2021              BasePy/2022
BaseCPU/2021                BaseGPU/2022              BasePy/2023
BaseCPU/2022                BaseGPU/2023              BasePy/2024
BaseCPU/2023                BaseGPU/2024              BasePy/2025(default)
BaseCPU/2024                BaseGPU/2025(default)     BaseR/2021
BaseCPU/2025(default)       BaseJDK/2021              BaseR/2022
BaseExtra/2021              BaseJDK/2022              BaseR/2023
BaseExtra/2022              BaseJDK/2023              BaseR/2024
BaseExtra/2023              BaseJDK/2024              BaseR/2025(default)
BaseExtra/2024              BaseJDK/2025(default)     BaseVEC/2023
BaseExtra/2025(default)     BaseJulia/2021            BaseVEC/2024
BaseGCC/2021                BaseJulia/2022            BaseVEC/2025(default)
```

② モジュールのロード

```
$ module load BaseCPU/2025
Loading compiler version 2023.2.4
Loading tbb version 2021.10.0
Loading compiler-rt version 2023.2.4
Loading oclfpga version 2023.2.4
  Load "debugger" to debug DPC++ applications with the gdb-oneapi debugger.
  Load "dpl" for additional DPC++ APIs: https://github.com/oneapi-src/oneDPL
Loading modulefiles version 2021.11
Loading advisor version 2023.2.0
Loading dal version 2023.2.0
```

```
Loading inspector version 2023.2.0
Loading intel_ipp_intel64 version 2021.9.0

For a full Intel(R) Integrated Performance Primitives functionality, you should load Intel(R)
oneAPI Compiler and Intel(R) oneAPI Threading Building Blocks modules.

Loading itac version 2021.10.0
Loading mkl version 2023.2.0
Loading vtune version 2023.2.0
Loading debugger version 2023.2.0
Loading dpl version 2022.2.0

Loading BaseCPU/2025
  Loading requirement: inteloneAPI/2023.2 tbb/latest compiler-rt/latest
    oclfpga/latest compiler/latest mpi/latest advisor/latest dal/latest
    inspector/latest intel_ipp_intel64/latest itac/latest mkl/latest
    vtune/latest debugger/latest dpl/latest
```

③ ロードされているモジュールの確認

```
$ module list
Currently Loaded Modulefiles:
 1) inteloneAPI/2023.2     9) inspector/latest
 2) tbb/latest            10) intel_ipp_intel64/latest
 3) compiler-rt/latest    11) itac/latest
 4) oclfpga/latest        12) mkl/latest
 5) compiler/latest       13) vtune/latest
 6) mpi/latest            14) debugger/latest
 7) advisor/latest        15) dpl/latest
 8) dal/latest            16) BaseCPU/2025(default)
```

④ 追加でロード可能なモジュール一覧の表示

```
$ module avail
--- /system/apps/rhel8/cpu/intel/inteloneAPI2023.2/2023.2.0/modulefiles ----
advisor/2023.2.0          dnnl-cpu-tbb/2023.2.0          itac/2021.10.0
advisor/latest            dnnl-cpu-tbb/latest            itac/latest
```

```
ccl/2021.10.0                dnnl/2023.2.0                    mkl/2023.2.0
ccl/latest                   dnnl/latest                      mkl/latest
compiler-rt/2023.2.4         dpct/2023.2.0                    mkl32/2023.2.0
compiler-rt/latest           dpct/latest                      mkl32/latest
compiler-rt32/2023.2.4       dpl/2022.2.0                     mpi/2021.10.0
compiler-rt32/latest         dpl/latest                       mpi/2021.11
compiler/2023.2.4            icc/2023.2.4                     mpi/latest
compiler/latest              icc/latest                       oclfpga/2023.2.0
compiler32/2023.2.4          icc32/2023.2.4                   oclfpga/2023.2.4
compiler32/latest            icc32/latest                     oclfpga/latest
dal/2023.2.0                 inspector/2023.2.0               tbb/2021.10.0
dal/latest                   inspector/latest                 tbb/latest
debugger/2023.2.0            intel_ipp_ia32/2021.9.0          tbb32/2021.10.0
debugger/latest              intel_ipp_ia32/latest            tbb32/latest
dev-utilities/2021.10.0      intel_ipp_intel64/2021.9.0       vtune/2023.2.0
dev-utilities/latest         intel_ipp_intel64/latest         vtune/latest
dnnl-cpu-gomp/2023.2.0       intel_ippcp_ia32/2021.8.0
dnnl-cpu-gomp/latest         intel_ippcp_ia32/latest
dnnl-cpu-iomp/2023.2.0       intel_ippcp_intel64/2021.8.0
dnnl-cpu-iomp/latest         intel_ippcp_intel64/latest

---------------------- /system/apps/env/cpu/Compiler ----------------------
intel/2020update4     inteloneAPI/2021.4    inteloneAPI/2023.2
inteloneAPI/2021.2    inteloneAPI/2023.0    inteloneAPI/2025.0

---------------- /system/apps/env/cpu/lib/inteloneAPI2023.0 ----------------
fftw/3.3.10          netcdf-c/4.9.2            pnetcdf-c/1.12.3
hdf5/1.12.3.mpi      netcdf-cxx/4.3.1          pnetcdf-cxx/1.12.3
hdf5/1.14.1          netcdf-fortran/4.6.1      pnetcdf-fortran/1.12.3

---------------------- /system/apps/env/cpu/devtool ----------------------
ArmForge/21.0    LinaroForge/24.1.1    XcalableMP/1.3.3

------------------------- /system/apps/env/Base -------------------------
BaseApp/2021                 BaseGCC/2022              BaseJulia/2023
```

BaseApp/2022	BaseGCC/2023	BaseJulia/2024
BaseApp/2023	BaseGCC/2024	BaseJulia/2025(default)
BaseApp/2024	BaseGCC/2025(default)	BasePy/2021
BaseApp/2025(default)	BaseGPU/2021	BasePy/2022
BaseCPU/2021	BaseGPU/2022	BasePy/2023
BaseCPU/2022	BaseGPU/2023	BasePy/2024
BaseCPU/2023	BaseGPU/2024	BasePy/2025(default)
BaseCPU/2024	BaseGPU/2025(default)	BaseR/2021
BaseCPU/2025(default)	BaseJDK/2021	BaseR/2022
BaseExtra/2021	BaseJDK/2022	BaseR/2023
BaseExtra/2022	BaseJDK/2023	BaseR/2024
BaseExtra/2023	BaseJDK/2024	BaseR/2025(default)
BaseExtra/2024	BaseJDK/2025(default)	BaseVEC/2023
BaseExtra/2025(default)	BaseJulia/2021	BaseVEC/2024
BaseGCC/2021	BaseJulia/2022	BaseVEC/2025(default)

⑤ 追加モジュールのロード

```
$ module load hdf5/1.14.1
```

module コマンドの詳細な内容は、オンラインドキュメント(man) あるいは、公式サイトのドキュメントをご参照ください。

- Modules v4.5.2 : Docs >> module
  https://modules.readthedocs.io/en/v4.5.2/module.html

3.2. HPC 用フロントエンドの利用方法

3.2.1. 利用準備

HPC フロントエンドにおける「NICE DCV(Desktop Cloud Visiualization)」の利用準備について説明します。以下の URL から「NICE DCV クライアント」を取得し、手元の PC へインストールします。Windows OS、Linux OS、macOS 版が用意されています。

https://www.nice-dcv.com/latest.html

3.2.2. NICE DCV サーバの仮想セッション作成

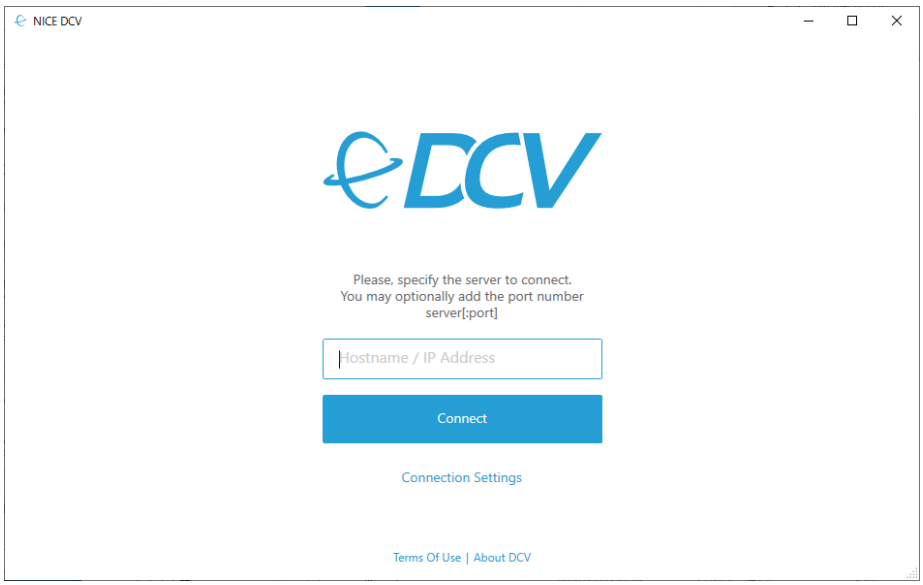
HPC フロントエンドへログインし、「NICE DCV サーバ」の仮想セッションを作成します。

```
$ dcv create-session --type=virtual セッション名
(仮想セッションを作成します)
$ dcv list-sessions
(作成した仮想セッションを確認します)
```

HPC フロントエンドは 4 台あります。仮想セッションを作成したフロントエンドのサーバに接続する必要があります。サーバ名およびセッション名は「3.2.3. NICE DCV クライアントの起動」で使用しますので、お手元に情報をお控えください。

3.2.3. NICE DCV クライアントの起動

「dcvviewer」を起動します。起動すると、以下のようなウィンドウが表示されます。



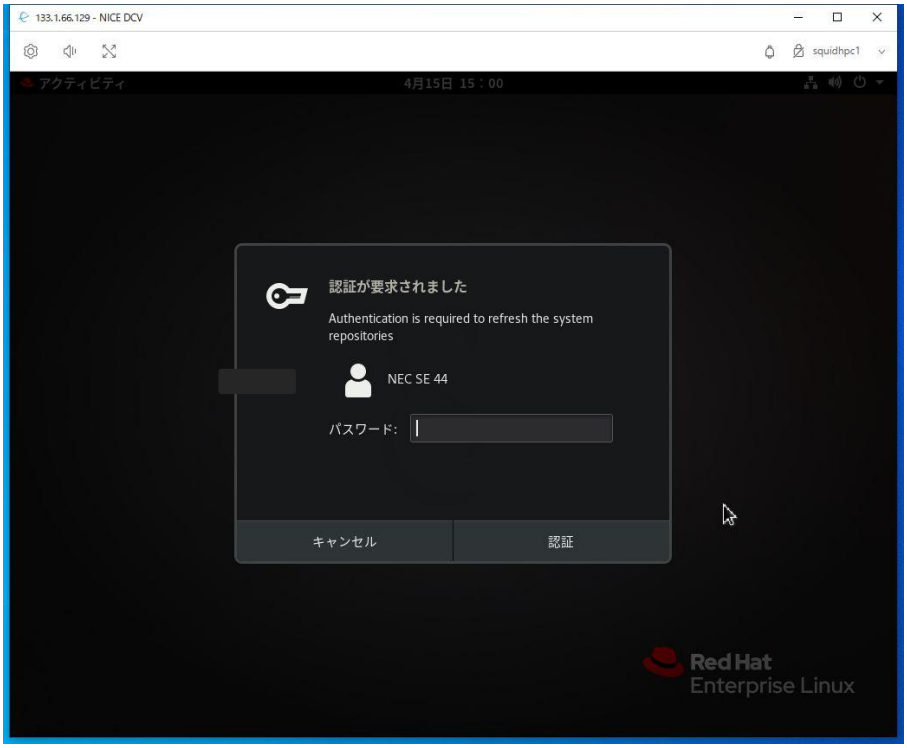
「3.2.2. NICE DCV サーバの仮想セッション作成」でセッションを作成したサーバに接続します。例えば、仮想セッションを作成したサーバが「squidhpc1」、セッション名が「mysession」であった

場合、「squidhpc1.hpc.cmc.osaka-u.ac.jp:5901/#mysession」と入力し、[Connect]をクリックします。

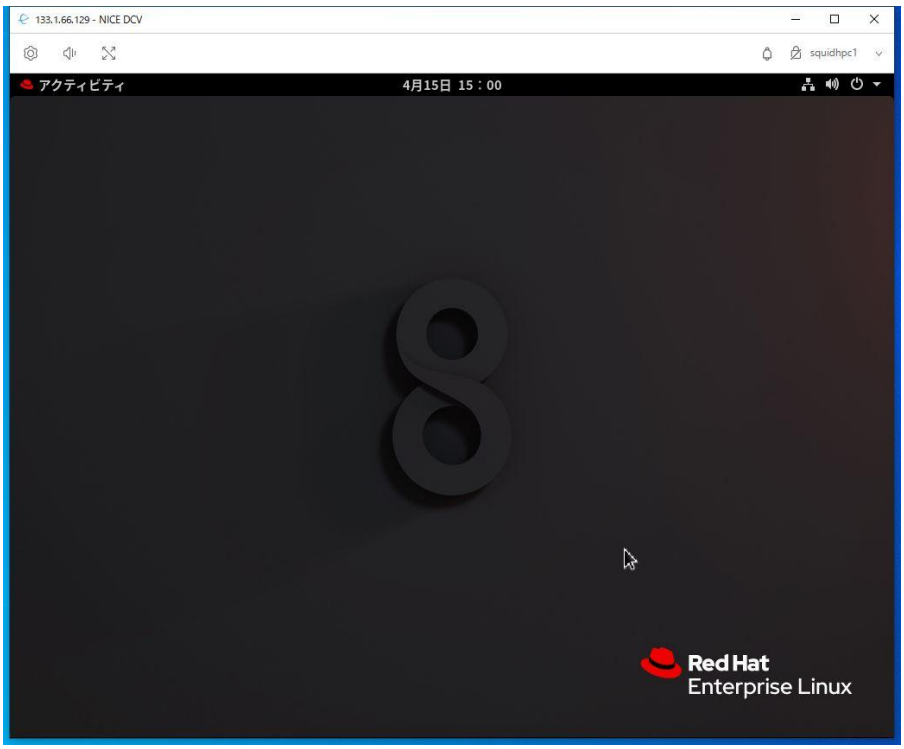
HPC フロントエンド上の「NICE DCV サーバ」へ接続が成功すると、以下のような画面が表示されます。ユーザ名、パスワードを入力し、[Login]をクリックします。

NICE DCV のログインに成功すると、以下のような HPC フロントエンドのデスクトップログイン画面が表示されます。

パスワードを入力し、[認証]をクリックします。



認証が成功すると、以下のような HPC フロントエンドのデスクトップ画面が表示されます。



3.2.4. NICE DCV サーバの仮想セッション削除

NICE DCV の利用が終了した場合、HPC フロントエンドへログインし、NICE DCV サーバの仮想セッションを削除します。

```
$ dcv close-session セッション名
(仮想セッションを削除します)
```

仮想セッションを作成したサーバに接続し削除する必要があります。

3.3. HPDA 用フロントエンドの利用方法

3.3.1. 利用準備

HPDA フロントエンドにおける、jupyter コンテナの利用準備について説明します。

- jupyter コンテナイメージの取得
  ローカルレジストリから、jupyter コンテナイメージの取得を行います。
  jupyter コンテナイメージを取得するには、任意のディレクトリで、以下のコマンドを実行します。

```
$ singularity pull jupyter.sif oras://cntm:5000/master_image/jupyter:2.0
```

※上記のコマンドで取得されるコンテナイメージの OS は Rocky Linux 8.6 (64bit)です。

取得が成功すると、実行したディレクトリ配下に jupyter.sif というファイルが作成されます。
jupyter.sif というファイルが存在する場合は、予め削除してから上記のコマンドを実行してください。※jupyter.sif というファイルが存在すると、コマンドの実行は失敗します。

3.3.2. jupyter コンテナの起動

HPDA フロントエンドにおける、jupyter コンテナの起動方法について説明します。

- jupyter コンテナ起動用シェルスクリプトの実行
  jupyter.sif のあるディレクトリで、jupyter コンテナ起動用シェルスクリプトを実行し、jupyter コンテナを起動します。

```
$ run_jupyter_container.sh
```

jupyter コンテナの起動に成功すると、ターミナルに以下のように表示されます。

```
$ run_jupyter_container.sh
jupyter Notebook URL https:// squidhpda1.hpc.cmc.osaka-u.ac.jp:10125 ①
INFO:    Converting SIF file to temporary sandbox...
INFO:    instance started successfully
jupyter login token : 3c4b9f3fb424b9059d1076c559adff837f9fc9ee8ce81842 ②
$
```

① jupyter Notebook URL

jypyter Notebook に Web ブラウザでアクセスする際の URL です。

ポート番号は、10000~11000 の範囲内で、空いている番号が自動で割り当てられます。

② jupyter login token

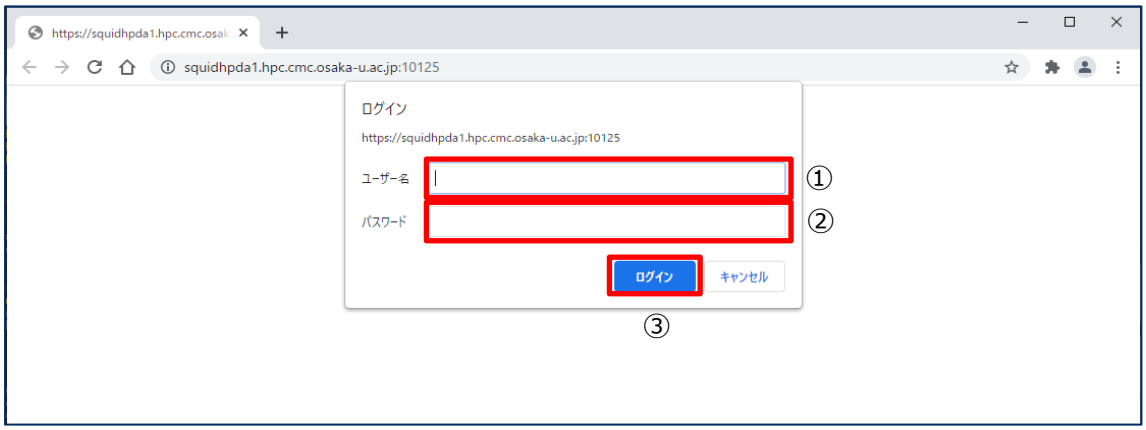
Web ブラウザよりアクセスした際、jupyter の認証画面で入力するトークンです。

3.3.3. jupyter コンテナへのアクセス

Web ブラウザより、jupyter コンテナにアクセスする方法について説明します。

- ユーザ名／パスワードによる認証

  Web ブラウザを起動し、「3.3.2 jupyter コンテナの起動」でターミナルに表示された、jupyter Notebook URL を指定します。



① ユーザ名

HPDA フロントエンドへのログイン時に使用した、ユーザ名を入力します。

② パスワード

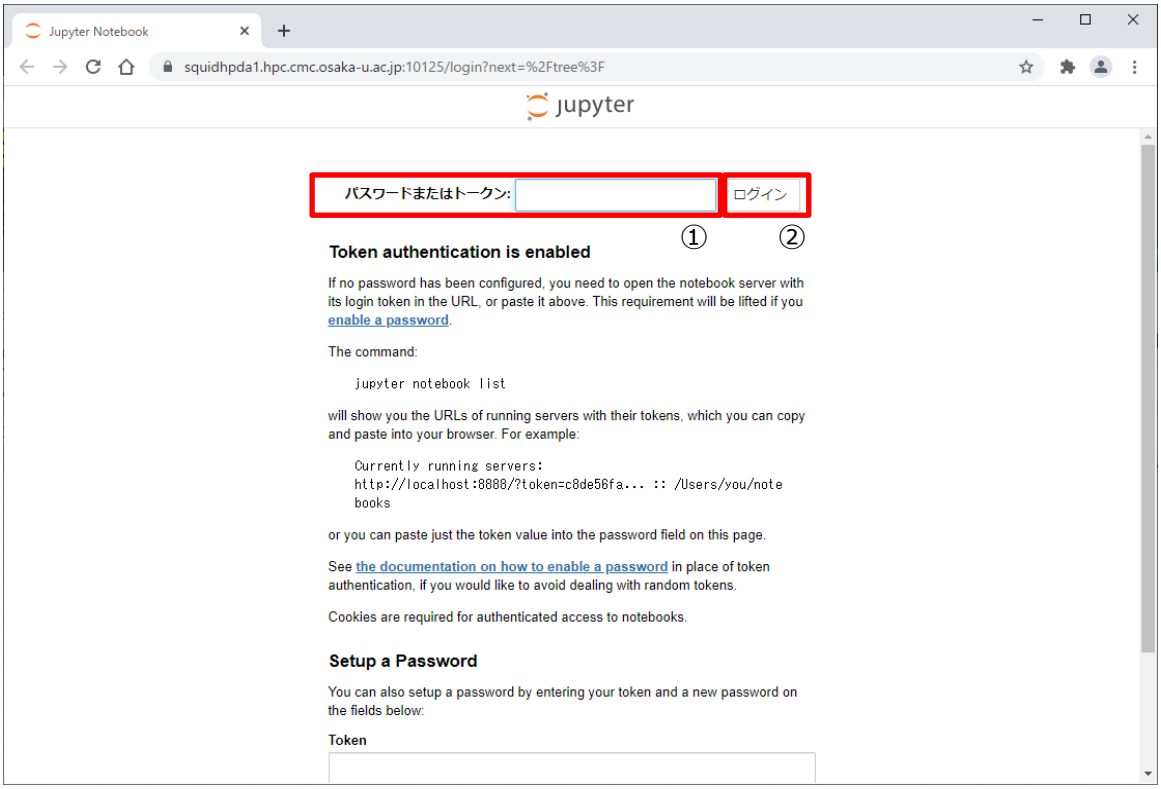
HPDA フロントエンドへのログイン時に使用した、パスワードを入力します。

③ ログイン

ログインボタンをクリックすると、ユーザ名／パスワードによる認証が実施されます。

- jupyter トークンよる認証

  ユーザ名／パスワードによる認証が成功すると、jupyter の認証画面が表示されます。



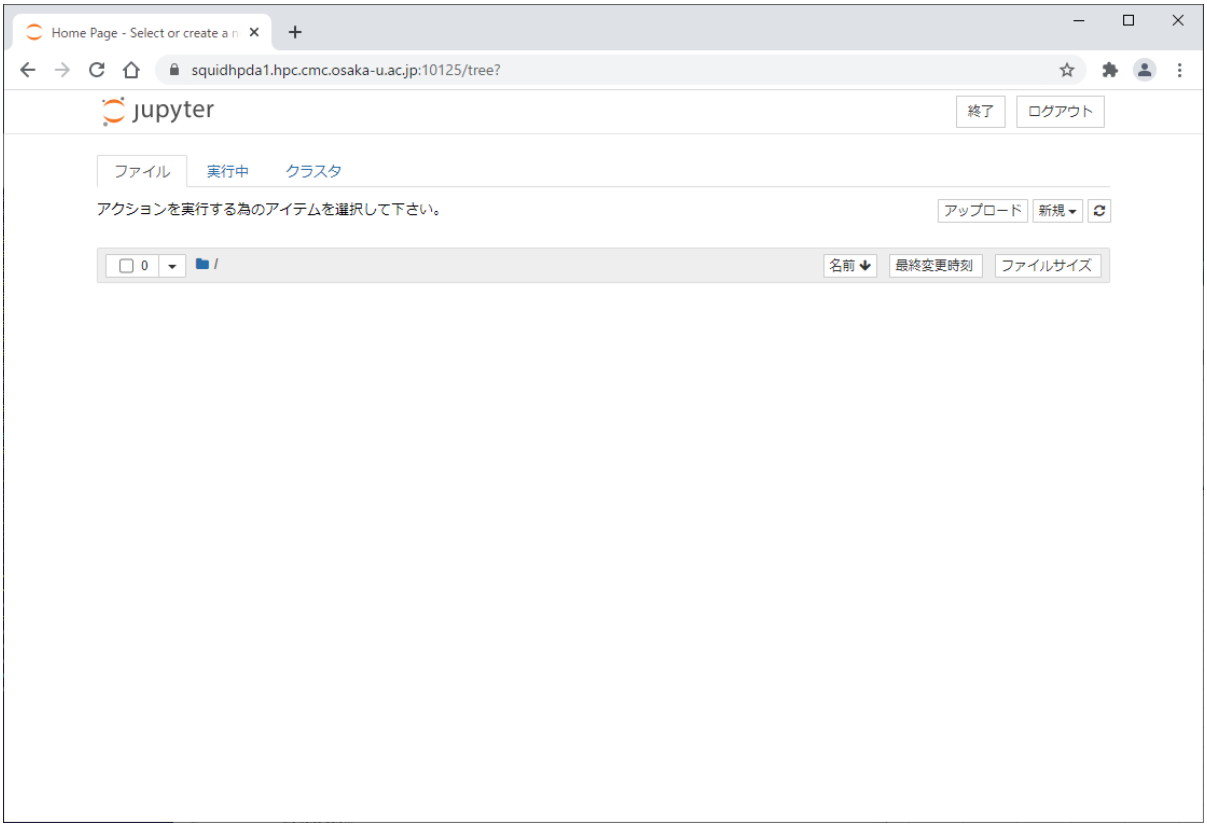
① パスワードまたはトークン

「3.3.2 jupyter コンテナの起動」でターミナルに表示された、jupyter login token を入力します。

② ログイン

ログインボタンをクリックします。

jupyter トークンによる認証が成功すると、jupyter Notebook の初期画面が表示されます。



3.3.4. jupyter コンテナのカスタマイズ

jupyter コンテナは利用者自身の好みに合わせてカスタマイズできるよう、ライブラリ等のコンポーネントはインストールしていない状態で提供しています。必要に応じて、ご自身でライブラリ等をインストールして利用できます。

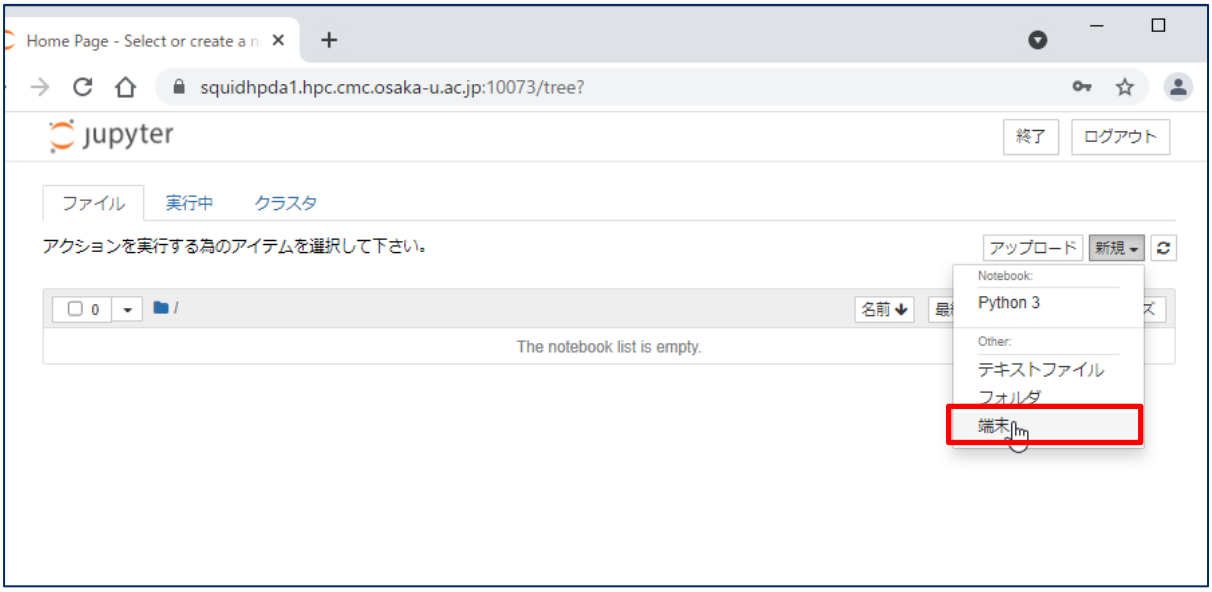
次に、jupyter コンテナに追加コンポーネントをインストールする方法について説明します。
ここでは、例として numpy、および matplotlib のインストール方法を紹介します。

- numpy のインストール

  「3.3.3 jupyter コンテナへのアクセス」で、jupyter Notebook の初期画面が表示された状態から作業を行います。

  ① 端末の起動

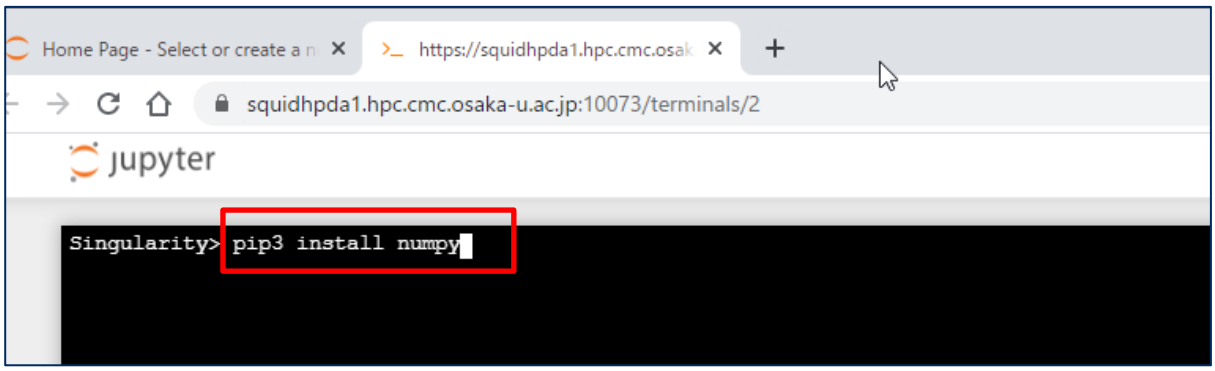
  初期画面右上の新規ボタンをクリックし、端末を選択します。



② pip コマンドの実行

端末に以下のコマンドを入力し、Enter キーを押下します。

コマンド：pip3 install numpy

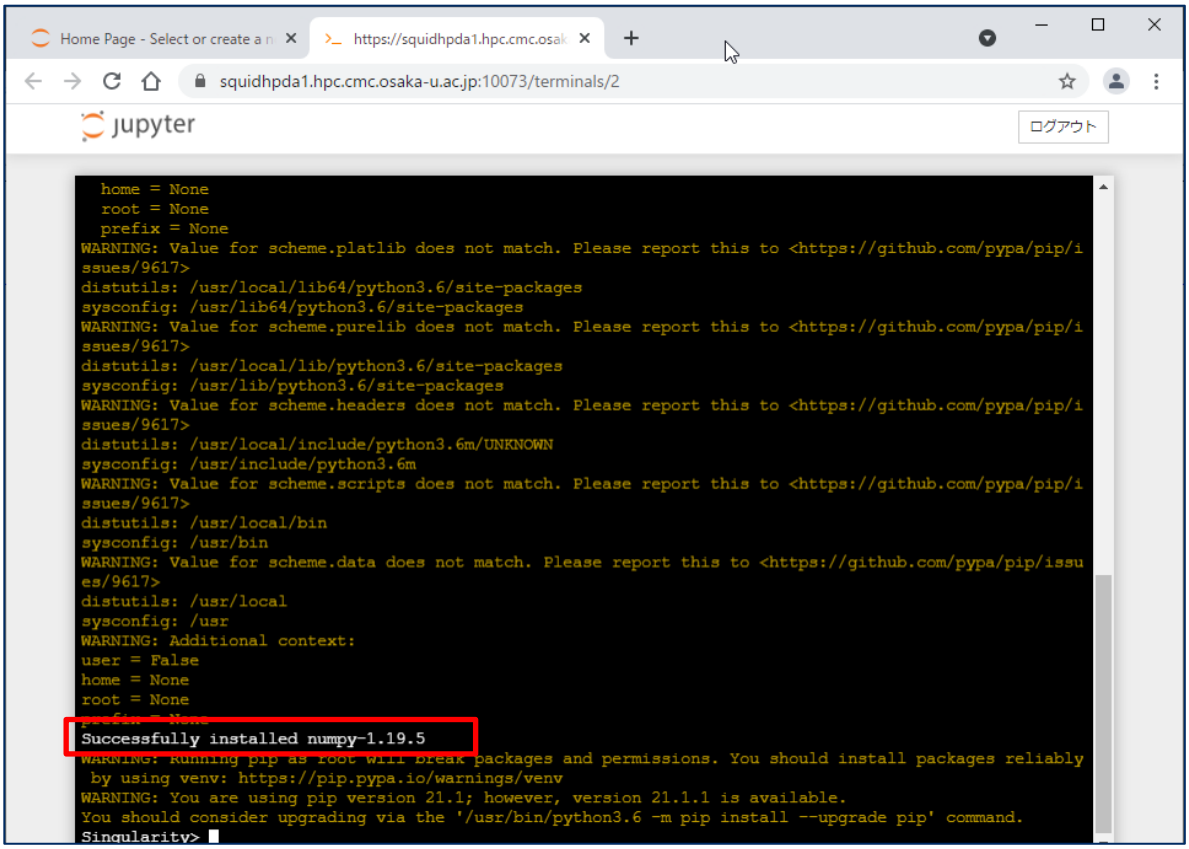


③ インストールの確認

以下のメッセージにより、インストールが成功したことを確認します。

メッセージ：Successfully installed numpy-x.x.x

※x.x.x の部分には、インストールされた numpy のバージョンが表示されます。

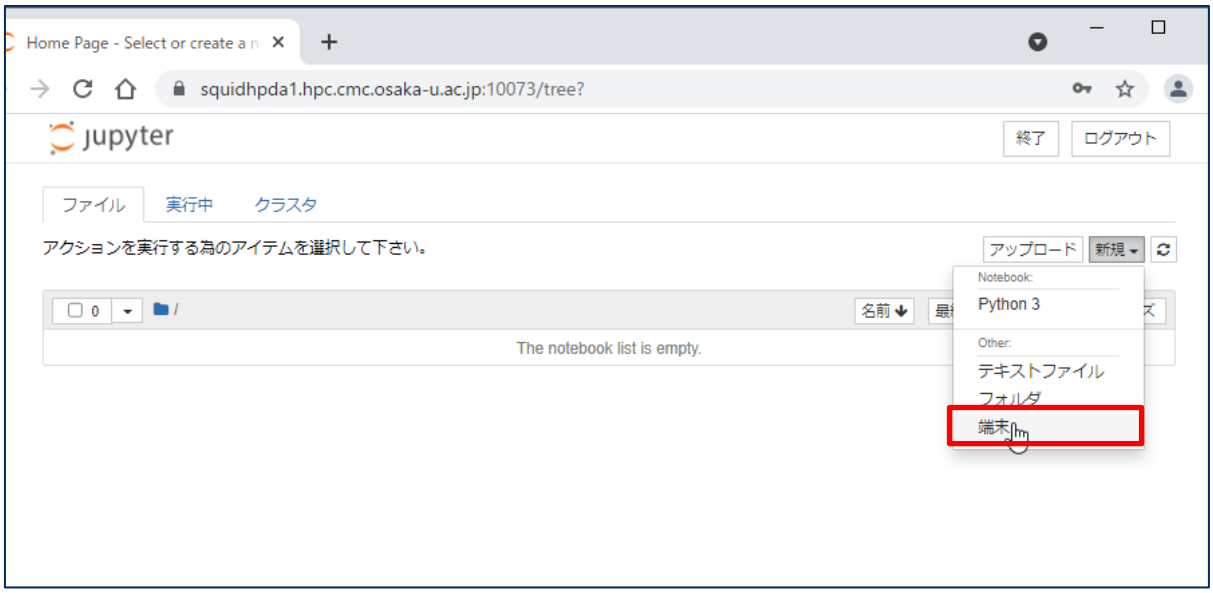


- matplotlib のインストール

  「3.3.3 jupyter コンテナへのアクセス」で、jupyter Notebook の初期画面が表示された状態から作業を行います。

  ① 端末の起動

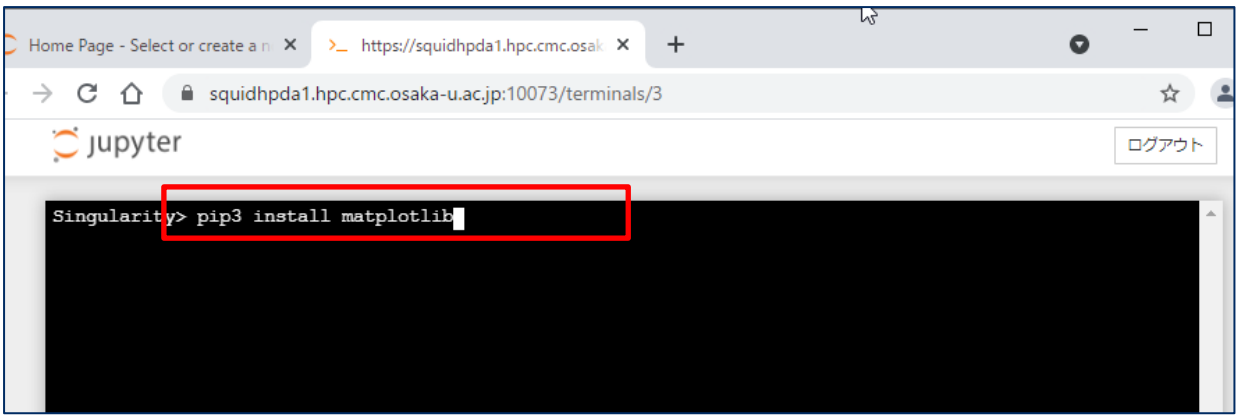
  初期画面右上の新規ボタンをクリックし、端末を選択します。



② pip コマンドの実行

端末に以下のコマンドを入力し、Enter キーを押下します。

コマンド：pip3 install matplotlib

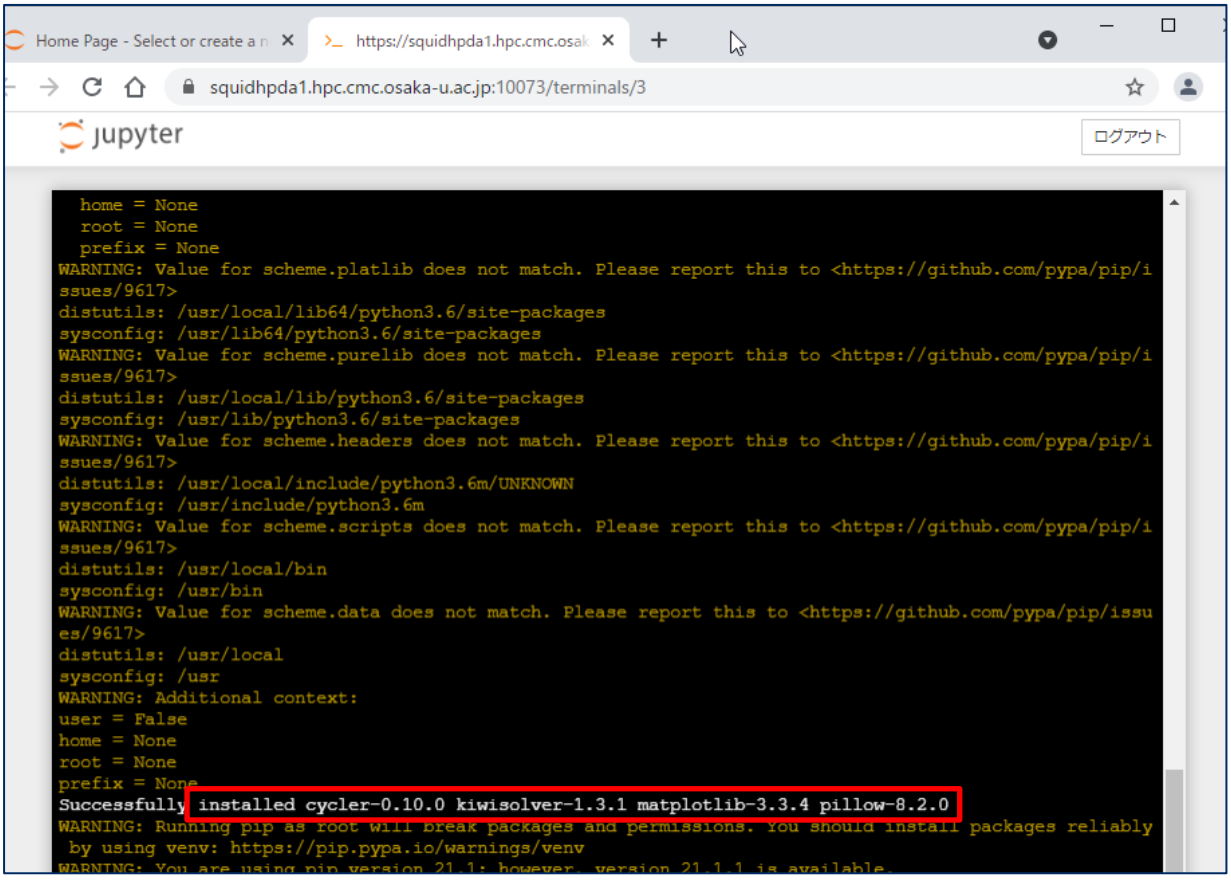


③ インストールの確認

以下のメッセージにより、インストールが成功したことを確認します。

メッセージ：Successfully installed cycler-x.x.x kiwisolver-x.x.x matplotlib-x.x.x pillow-x.x.x

※x.x.x の部分には、インストールされた matplotlib、および依存パッケージのバージョンが表示されます。



3.3.5. jupyter コンテナの停止

HPDA フロントエンドにおける、jupyter コンテナの停止方法について説明します。
jupyter コンテナ内の変更内容は、jupyter.sif に自動で反映されます。

- jupyter コンテナの停止
  「3.3.2 jupyter コンテナの起動」で jupyter を起動したディレクトリで、jupyter コンテナ停止用シェルスクリプトを実行し、jupyter コンテナを停止します。

```
$ stop_jupyter_container.sh
```

jupyter コンテナの停止に成功すると、ターミナルに以下のように表示されます。

```
$ stop_jupyter_container.sh
INFO:   Starting build...
INFO:   Creating SIF file...
INFO:   Build complete: ./jupyter_tmp.sif
INFO:   Stopping jupyter instance of /var/tmp/rootfs-940268085/root (PID=19101)
$
```

3.3.6. julia、R カーネルインストール済み jupyter コンテナの利用

julia、または R カーネルをインストール済みの jupyter コンテナの利用方法について説明します。

基本的な利用方法は標準の jupyter コンテナと同じですが、コンテナイメージの取得と起動／停止方法が異なります。

- コンテナイメージの取得

① Julia カーネルインストール済み jupyter コンテナ

```
$ singularity pull jupyter-julia.sif oras://cntm:5000/master_image/jupyter-julia:2.0
```

※上記のコマンドで取得されるコンテナイメージの OS は Rocky Linux 8.6 (64bit)です。

② R カーネルインストール済み jupyter コンテナ

```
$ singularity pull jupyter-r.sif oras://cntm:5000/master_image/jupyter-r:2.0
```

※上記のコマンドで取得されるコンテナイメージの OS は Rocky Linux 8.6 (64bit)です。

- コンテナの起動

① Julia カーネルインストール済み jupyter コンテナ

```
$ run_jupyter_container.sh -k julia
```

または

```
$ run_jupyter_container.sh --kernel julia
```

② R カーネルインストール済み jupyter コンテナ

```
$ run_jupyter_container.sh -k R
```

または

```
$ run_jupyter_container.sh --kernel R
```

- コンテナの停止

① Julia カーネルインストール済み jupyter コンテナ

```
$ stop_jupyter_container.sh -k julia
```

または

```
$ stop_jupyter_container.sh --kernel julia
```

② R カーネルインストール済み jupyter コンテナ

```
$ stop_jupyter_container.sh -k R
```

または

```
$ stop_jupyter_container.sh --kernel R
```

4. プログラム開発

4.1. 共通事項

本システムでは、プログラムの作成から、作成モジュールの実行までを、HPC フロントエンドまたは HPDA フロントエンドで行います。基本的なプログラムの実行までの流れは以下となります。本項では「② ソースコードから実行モジュールの生成（コンパイル）」の方法について詳細を記述します。

① ソースコードの作成・編集・修正

② ソースコードから実行モジュールの生成（コンパイル）

③ 実行モジュールの実行

4.1.1. コンパイラ・MPI の利用

本システムでは、各種のプログラミング言語が利用可能です。プログラミング言語を使うためのコンパイラや MPI、付随するライブラリの環境セットは、ベース環境としてまとめられています。本節ではプログラミング言語ごとのベース環境の概要を記載します。

(1)C/C++言語、FORTRAN 言語

C/C++言語、FORTRAN 言語のプログラム開発に際しては、汎用 CPU、ベクトル、GPGPU の各計算環境に適したコンパイラを利用できます。また、スレッド並列、MPI 並列による並列実行モジュールの作成も可能です。

ご利用の際は、「3.1.3.環境設定」に従い、「Environment Modules」を利用して、モジュールを読み込みます。モジュールの中には、各計算環境でプログラム開発を行うのに適した構成を、推奨環境として整備しています。各推奨環境の構成は以下の通りです。

計算環境	推奨環境 モジュール名	コンパイラ	MPI
汎用 CPU 計算環境	BaseCPU	Intel Parallel Studio Intel oneAPI	Intel MPI
ベクトル計算環境	BaseVEC	NEC SDK for VE	NEC MPI
GPGPU 計算環境	BaseGPU	NVIDIA HPC SDK CUDA	Open MPI
(なし)	BaseGCC	GNU Compiler	Open MPI

※ 汎用 CPU 計算環境で BaseGPU の利用など、推奨環境と異なるモジュールを利用することも可

能です。ただし、推奨環境の方が、より計算環境に適したプログラム開発を行えます。

具体的な手順の例としては、汎用 CPU 計算環境において推奨環境を利用する場合、以下のようにモジュールをロードします。

```
$ module avail
------------------------------ /system/apps/env/Base ------------------------------
BaseApp/2021                BaseGCC/2022              BaseJulia/2023
BaseApp/2022                BaseGCC/2023              BaseJulia/2024
BaseApp/2023                BaseGCC/2024              BaseJulia/2025(default)
BaseApp/2024                BaseGCC/2025(default)   BasePy/2021
BaseApp/2025(default)     BaseGPU/2021              BasePy/2022
BaseCPU/2021                BaseGPU/2022              BasePy/2023
BaseCPU/2022                BaseGPU/2023              BasePy/2024
BaseCPU/2023                BaseGPU/2024              BasePy/2025(default)
BaseCPU/2024                BaseGPU/2025(default)   BaseR/2021
BaseCPU/2025(default)     BaseJDK/2021              BaseR/2022
BaseExtra/2021              BaseJDK/2022              BaseR/2023
BaseExtra/2022              BaseJDK/2023              BaseR/2024
BaseExtra/2023              BaseJDK/2024              BaseR/2025(default)
BaseExtra/2024              BaseJDK/2025(default)   BaseVEC/2023
BaseExtra/2025(default)   BaseJulia/2021           BaseVEC/2024
BaseGCC/2021                BaseJulia/2022           BaseVEC/2025(default)

$ module load BaseCPU/2025
```

推奨環境の設定や各コンパイルコマンド、並列プログラムのコンパイル方法の詳細は以降の章で記述します。

- 汎用 CPU 計算環境用コンパイラ
  「4.2 汎用 CPU 計算環境向けプログラムのコンパイル」をご参照ください。

- ベクトル計算環境用コンパイラ
  「4.4 ベクトル計算環境向けプログラム」をご参照ください。

- GPGPU 計算環境用コンパイラ

「4.5 GPGPU 計算環境向けプログラムのコンパイル」をご参照ください。

- GNU コンパイラ

  「4.5 GPGPU 計算環境向けプログラムのコンパイル」をご参照ください。

(2)その他のプログラミング言語

C/C++言語、FORTRAN 言語以外のプログラム言語についても、ベース環境を用意しています。利用可能なプログラミング言語環境は以下の通りです。

言語環境	モジュール名	バージョン	説明	備考
Python	BasePy	3.6	Python 言語向け	Python2 も利用可
R	BaseR	4.0.3	R 言語向け	
JAVA	BaseJDK	11	JAVA 言語向け	OpenJDK
Julia	BaseJulia	1.6.1	Julia 言語向け	

これらのプログラム言語環境の起動方法を以下に示します。

A) Python

フロントエンドにログイン後以下を実行します。

```
$ module load BasePy
$ python3
```

B) R

フロントエンドにログイン後以下を実行します。

```
$ module load BaseR
$ R
```

C) JAVA

フロントエンドにログイン後以下を実行します。

```
$ module load BaseJDK
$ java --version
```

D) Julia

フロントエンドにログイン後以下を実行します。

```
$ module load BaseJulia
```

```
$ julia
```

近年データ分析用途などで、利用が増えてきている Python 言語に関して、「4.8 Python の利用方法」に詳細を記載します。

- Python 言語
  「4.8 Python の利用方法」をご参照ください。

4.1.2. ライブラリの利用

本システムで利用可能なライブラリ群は module コマンドにより環境変数を設定します。

また、各ライブラリのモジュールは<モジュール名>/<バージョン>でバージョン管理されます。ライブラリは、各ベース環境に同梱されています。それらのライブラリはベースモジュールを読み込んだ後に利用可能となります。

利用可能なライブラリ、言語モジュールは以下の通りです。

ライブラリ 言語モジュール	モジュール名					
	CPU	VEC	GPU	GCC	Py	R
Intel MKL (※)	○	-	-	-	-	-
NEC Numeric Library Collection (※)	-	○	-	-	-	-
GNU Scientific Library	-	-	-	○	-	-
HDF5	○	○	○	-	-	-
NetCDF	○	○	○	-	-	-
Parallel netcdf	○	○	○	-	-	-
Keras	-	-	-	-	○	-
PyTorch	-	-	-	-	○	-
TensorFlow	-	-	-	-	○	-
pbdR	-	-	-	-	-	○

※ BLAS, LAPACK, ScaLAPACK は「NEC Numeric Library Collection」と「Intel MKL」に含まれています。

ライブラリの利用方法の具体的な内容は、以下の各節の「ライブラリの利用」の項に記載します。適宜ご参照ください。

「4.2 汎用 CPU 計算環境向けプログラムのコンパイル」

「4.4 ベクトル計算環境向けプログラムのコンパイル」

「4.5 GPGPU 計算環境向けプログラムのコンパイル」

4.1.3. 開発支援ツールの使用方法

本システムでは、以下の開発支援ツールを備えています。これらの開発支援ツールは、インタラクティブキューにログインして使用します。

開発支援ツール	汎用 CPU 計算環境	ベクトル計算環境	GPGPU 計算環境
Linaro Forge DDT/MAP	○	-	○
NEC Parallel Debugger	-	○	-

- Linaro Forge DDT/MAP の起動方法

  「Linaro Forge」は、Linux 上の C/C++/FORTRAN アプリケーションのデバッグ、プロファイラ、最適化ツールなどを備えた開発支援ツールスイートです。デバッガである「Linaro DDT」ならびに、並列アプリケーション対応のプロファイラである「Linaro MAP」が主要コンポーネントとなります。次の手順で起動します。

```
$ module load BaseCPU/2025
$ module load LinaroForge/24.1.1
$ ddt
```

- NEC Parallel Debugger の起動方法

  「NEC Parallel Debugger」は Vector Engine 用のデバッグツールで、Eclipse PTP (Parallel Tools Platform)に対する MPI アプリケーションのデバッグ用プラグインを提供しています。「NEC Parallel Debugger」は、次の手順で起動します。

  (1) フロントエンド環境にログインするサーバ上で事前に X サーバ(Xming、VcXsrv 等)を起動します。

  (2) フロントエンド環境に X forwarding を有効にしてログインします。

  (3) Eclipse PTP を起動します。

```
$ module load BaseVEC/2025
$ module load EclipsePTP/4.7.3a
$ eclipse
```

(4) Eclipse Launcher のウィンドウが開くので、Workspace ディレクトリ（デバッグ対象のアプリケーションの make 環境を格納するディレクトリ）を入力して「Launch」をクリックします。

使用方法の詳細については NEC Aurora Forum（https://www.hpc.nec）の以下のページをご参照ください。

Aurora Forum（https://www.hpc.nec）
メニュー ＞ ドキュメント ＞ NEC SDK ＞ NEC Parallel Debugger ユーザーズガイド

4.2. 汎用 CPU 計算環境向けプログラムのコンパイル(Intel one API/Intel Parallel Studio)

汎用 CPU 計算環境向けプログラムのコンパイル方法を記載します。ここでは、推奨環境である BaseCPU 上でのコンパイル方法を記載します。プログラムコンパイル時には、BaseCPU 環境を事前に読み込んでください。

```
$ module load BaseCPU/2025
```

※Intel Parallel Studio 利用の場合は BaseCPU/2021

4.2.1. シリアル実行

コンパイルは、フロントエンド上で実施します。

各プログラム言語のコンパイルコマンドは以下になります。

プログラム言語	Intel one API / Intel Parallel Studio
FORTRAN	ifort
C	icc
C++	icpc

以下はコンパイル例です。”-o”オプションにより、作成する実行ファイル名を指定できます。指定しなかった場合、ファイル名が a.out で作成されます。

最初に module コマンドで Intel コンパイラの環境変数設定モジュールをロードする必要があります。BaseCPU モジュールをロードすると自動的に、標準バージョンの Intel コンパイラがロードされます。

```
$ module load BaseCPU/2025

$ ifort -o sample sample.f90
$ icc -o sample sample.c
$ icpc -o sample sample.cpp
```

4.2.2. スレッド並列実行

スレッド並列で実行する場合は、コンパイル時に以下のオプションを指定して下さい。

並列モデル	Intel one API / Intel Parallel Studio
OpenMP	-qopenmp
自動並列	-parallel

実行スレッド数は、実行時に環境変数 OMP_NUM_THREADS に<実行スレッド数>を指定して定義します。

以下はコンパイル例です。最初に module コマンドで Intel コンパイラの環境変数設定モジュールをロードする必要があります。

```
$ module load BaseCPU/2025

$ ifort -o sample_omp -qopenmp sample_omp.f90
$ ifort -o sample_smp -parallel sample_smp.f90
```

4.2.3. MPI 並列実行

MPI 環境は Intel MPI をご利用いただけます。

コンパイルコマンドは以下の通りです。

プログラム言語	IntelMPI の場合
FORTRAN	mpiifort
C	mpiicc
C++	mpiicpc

以下はコンパイル例です。

```
$ module load BaseCPU/2025

$ mpiifort -o sample_mpi sample_mpi.f90
$ mpiicc   -o sample_mpi sample_mpi.c
$ mpiicpc -o sample_mpi sample_mpi.cpp
```

4.2.4. ライブラリの利用

汎用 CPU 計算環境向けプログラムにおけるライブラリの利用方法を記載します。

(1) Intel Math Kernel Library (Intel MKL)

Intel Math Kernel Library は Intel 社が開発している数値演算ライブラリです。BLAS、LAPACK、ScaLAPACK 等を含む多様なライブラリの利用が可能です。

本システムでは、BaseCPU モジュールで読み込まれる Intel 社製コンパイラの環境で利用可能です。以下に本システムでの、ライブラリのリンク例を記載します。

```
# シリアル実行で MKL をリンク
```

```
$ module load BaseCPU/2025
$ icc -mkl=sequential sample.c                  # 動的リンク
$ icc -mkl=sequential -static-intel sample.c    # 静的リンク

# スレッド並列実行で MKL をリンク
$ module load BaseCPU/2025
$ icc -mkl=parallel sample.c                    # 動的リンク
$ icc -mkl=parallel -static-intel sample.o      # 静的リンク

# MPI 並列実行で MKL をリンク
$ module load BaseCPU/2025
$ mpiicc -mkl=cluster sample.c                  # 動的リンク
$ mpiicc -mkl=cluster -static-intel sample.c    # 静的リンク
```

Intel MKL の詳細な利用方法は、公式のオンラインドキュメントをご参照ください。

- Modul Intel Math Kernel Library – Documentation
  https://software.intel.com/content/www/us/en/develop/articles/intel-math-kernel-library-documentation.html

(2) HDF5

HDF5 は、大量データを階層的に構造化して格納することが可能なファイルを生成するためのライブラリです。計算環境毎に整備されています。このライブラリを利用するためには、コンパイル時に-lhdf5 のオプションを付与します。以下はライブラリのリンク例です。

```
$ module load BaseCPU
$ module load hdf5
$ icc -o h5-sample h5-sample.c -lhdf5
```

(3) NetCDF

NetCDF は、科学的な多次元データを格納する目的で、共通データ形式として広く使われているバイナリファイルフォーマットが利用できるライブラリです。本システムでは、計算環境毎に、C 言語/C++言語/FORTRAN 言語のそれぞれから利用できるように整備されています。ライブラリを利用するためには、コンパイル時に-lnetcdf 等 のオプションを付与します。以下はライブラリのリンク例です。

・C 言語

```
$ module load BaseCPU
$ module load netcdf-c
$ icc -o ncdf-sample ncdf-sample.c -lnetcdf
```

・C++言語

```
$ module load BaseCPU
$ module load netcdf-cxx
$ icpc -o ncdf-sample ncdf-sample.cpp -lnetcdf_c++4 -lnetcdf
```

・FORTRAN 言語

```
$ module load BaseCPU
$ module load netcdf-fortran
$ ifort -o ncdf-sample ncdf-sample.f90 -lnetcdff -lnetcdf
```

(4) PnetCDF

PnetCDF は、NetCDF 形式のファイルに対して、並列プログラムから同時アクセスを行うためのライブラリです。本システムでは、計算環境毎に、C 言語/C++言語/FORTRAN 言語のそれぞれから利用できるように整備されています。ライブラリを利用するためには、コンパイル時に-lpnetcdf 等のオプションを付与します。以下はライブラリのリンク例です。

・C 言語

```
$ module load BaseCPU
$ module load pnetcdf-c
$ mpiicc -o pncdf-sample pncdf-sample.c -lpnetcdf
```

・C++言語

```
$ module load BaseCPU
$ module load pnetcdf-cxx
$ mpiicpc -o pncdf-sample pncdf-sample.cpp -lpnetcdf
```

・FORTRAN 言語

```
$ module load BaseCPU
$ module load pnetcdf-fortran
```

```
$ mpiifort -o pncdf-sample pncdf-sample.f90 -lpnetcdf
```

4.3. 汎用 CPU 計算環境向けプログラムのコンパイル（Intel one API(2025 年度以降)）

汎用 CPU 計算環境向けプログラムのコンパイル方法を記載します。ここでは、推奨環境である BaseCPU 上でのコンパイル方法を記載します。プログラムコンパイル時には、Intel コンパイラの環境変数設定モジュールを事前に読み込んでください。

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
```

4.3.1. シリアル実行

コンパイルは、フロントエンド上で実施します。

各プログラム言語のコンパイルコマンドは以下になります。

プログラム言語	Intel one API
FORTRAN	ifx
C	icx
C++	icpx

以下はコンパイル例です。”-o”オプションにより、作成する実行ファイル名を指定できます。指定しなかった場合、ファイル名が a.out で作成されます。

最初に module コマンドで Intel コンパイラの環境変数設定モジュールをロードする必要があります。

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0

$ ifx -o sample sample.f90
$ icx -o sample sample.c
$ icpx -o sample sample.cpp
```

4.3.2. スレッド並列実行

スレッド並列で実行する場合は、コンパイル時に以下のオプションを指定して下さい。

並列モデル	Intel one API
OpenMP	-qopenmp

実行スレッド数は、実行時に環境変数 OMP_NUM_THREADS に<実行スレッド数>を指定して定義します。

以下はコンパイル例です。最初に module コマンドで Intel コンパイラの環境変数設定モジュールをロードする必要があります。

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0

$ ifx -o sample_omp -qopenmp sample_omp.f90
```

4.3.3. MPI 並列実行

MPI 環境は Intel MPI をご利用いただけます。

コンパイルコマンドは以下の通りです。

プログラム言語	IntelMPI の場合
FORTRAN	mpiifx
C	mpiicx
C++	mpiicpx

以下はコンパイル例です。

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0

$ mpiifx -o sample_mpi sample_mpi.f90
$ mpiicx   -o sample_mpi sample_mpi.c
$ mpiicpx -o sample_mpi sample_mpi.cpp
```

4.3.4. ライブラリの利用

汎用 CPU 計算環境向けプログラムにおけるライブラリの利用方法を記載します。

(1) Intel Math Kernel Library (Intel MKL)

Intel Math Kernel Library は Intel 社が開発している数値演算ライブラリです。BLAS、LAPACK、ScaLAPACK 等を含む多様なライブラリの利用が可能です。

本システムでは、Intel 社製コンパイラの環境で利用可能です。以下に本システムでの、ライブラリのリンク例を記載します。

```
# シリアル実行で MKL をリンク

$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
$ icx -qmkl=sequential sample.c                  # 動的リンク
$ icx -qmkl=sequential -static-intel sample.c    # 静的リンク
```

```
# スレッド並列実行で MKL をリンク
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
$ icx -qmkl=parallel sample.c                    # 動的リンク
$ icx -qmkl=parallel -static-intel sample.o      # 静的リンク

# MPI 並列実行で MKL をリンク
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
$ mpiicx -qmkl=cluster sample.c                  # 動的リンク
$ mpiicx -qmkl=cluster -static-intel sample.c    # 静的リンク
```

Intel MKL の詳細な利用方法は、公式のオンラインドキュメントをご参照ください。

- Modul Intel Math Kernel Library – Documentation
  https://software.intel.com/content/www/us/en/develop/articles/intel-math-kernel-library-documentation.html

(2) HDF5

HDF5 は、大量データを階層的に構造化して格納することが可能なファイルを生成するためのライブラリです。計算環境毎に整備されています。このライブラリを利用するためには、コンパイル時に-lhdf5 のオプションを付与します。以下はライブラリのリンク例です。

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
$ module load hdf5
$ icx -o h5-sample h5-sample.c -lhdf5
```

(3) NetCDF

NetCDF は、科学的な多次元データを格納する目的で、共通データ形式として広く使われているバイナリファイルフォーマットが利用できるライブラリです。本システムでは、計算環境毎に、C 言語/C++言語/FORTRAN 言語のそれぞれから利用できるように整備されています。ライブラリを利用するためには、コンパイル時に-lnetcdf 等 のオプションを付与します。以下はライブラリのリンク例です。

・C 言語

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
$ module load netcdf-c
$ icx -o ncdf-sample ncdf-sample.c -lnetcdf
```

・C++言語

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
$ module load netcdf-cxx
$ icpx -o ncdf-sample ncdf-sample.cpp -lnetcdf_c++4 -lnetcdf
```

・FORTRAN 言語

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
$ module load netcdf-fortran
$ ifx -o ncdf-sample ncdf-sample.f90 -lnetcdff -lnetcdf
```

(4) PnetCDF

PnetCDF は、NetCDF 形式のファイルに対して、並列プログラムから同時アクセスを行うためのライブラリです。本システムでは、計算環境毎に、C 言語/C++言語/FORTRAN 言語のそれぞれから利用できるように整備されています。ライブラリを利用するためには、コンパイル時に-lpnetcdf 等のオプションを付与します。以下はライブラリのリンク例です。

・C 言語

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
$ module load pnetcdf-c
$ mpiicx -o pncdf-sample pncdf-sample.c -lpnetcdf
```

・C++言語

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
$ module load pnetcdf-cxx
$ mpiicpx -o pncdf-sample pncdf-sample.cpp -lpnetcdf
```

・FORTRAN 言語

```
$ module load /system/apps/env/cpu/Compiler/inteloneAPI/2025.0
$ module load pnetcdf-fortran
$ mpiifx -o pncdf-sample pncdf-sample.f90 -lpnetcdf
```

4.4. ベクトル計算環境向けプログラムのコンパイル

ベクトル計算環境向けプログラムのコンパイル方法を記載します。ここでは、推奨環境である BaseVEC 上でのコンパイル方法を記載します。プログラムコンパイル時には、BaseVEC 環境を事前に読み込んでください。

```
$ module load BaseVEC/2025
```

4.4.1. シリアル実行

ベクトル計算環境の演算器である Vector Engine 向けの実行モジュール生成には、NEC コンパイラを使用します。コンパイルコマンドは以下になります。フロントエンド上でコンパイルを実行します。

プログラミング言語	NEC SDK for VE
Fortran	nfort
C	ncc
C++	nc++

主なコンパイルオプションは以下になります。詳細なオプション情報と環境変数情報は、man コマンドにて確認ください。

オプション	機能
-On (n=0~4)	最適化オプション 0：最適化、自動ベクトル化、並列化、インライン展開を適用しない 1：副作用を伴わない最適化・自動ベクトル化を適用する 2：副作用を伴う最適化・自動ベクトル化を適用する(既定値) 3：副作用を伴う最適化・自動ベクトル化、および、多重ループの最適化を適用する 4：言語仕様を逸脱した副作用を伴う最大限の最適化・自動ベクトル化を適用する
-o	作成する実行ファイル名を指定する
-mparallel	自動並列を有効にする 実行スレッド数は実行時に環境変数 OMP_NUM_THREADS に指定
-fopenmp	OpenMP 機能を有効にする 実行スレッド数は実行時に環境変数 OMP_NUM_THREADS に指定

-finline-functions	自動インライン展開を行う
-p	プロファイラを有効にする 実行後に性能測定結果ファイル gmon.out が出力 ngprof コマンドで gmon.out の内容を表示
-proginf	PROGINF 機能用の実行ファイルを生成する (既定値) 実行時に環境変数 VE_PROGINF=YES(または DETAIL)がセットされているとき PROGINF 情報を出力する
-ftrace	FTRACE 機能用のオブジェクトファイル、及び実行ファイルを生成する
-traceback	実行時に環境変数VE_TRACEBACKがセットされているとき、トレースバック情報を出力するオブジェクトファイル、実行ファイルを生成する
-g	バックトレースの出力に必要な行番号情報など最小限の情報を生成する
-report-all	診断メッセージリストと編集リストを出力する
-report-diagnostics	診断メッセージリストを出力する
-report-format	編集リストを出力する

以下は、NEC コンパイラを利用して、各プログラム言語をコンパイルする例になります。最初に BaseVEC のモジュールをロードした後、各種コンパイルコマンドを実行します。

```
$ module load BaseVEC/2025

$ nfort -o sample sample.f90
$ ncc -o sample sample.c
$ nc++ -o sample sample.cpp
```

4.4.2. スレッド並列実行

NEC コンパイラでは自動並列と OpenMP 並列の 2 つのスレッド並列実行が利用できます。

自動並列の場合はコンパイル時に"-mparallel"を指定し、OpenMP 並列の場合はコンパイル時に"-fopenmp"を指定します。スレッド数は実行時に環境変数 OMP_NUM_THREADS に指定します。

例 1) 自動並列による 8 スレッド実行

```
$ module load BaseVEC/2025
$ nfort -mparallel test.f                    # 自動並列を有効にしてコンパイル
```

```
$ export OMP_NUM_THREADS=8          # 実行スレッド数を指定（※）
$ ./a.out                           # 実行ファイルを実行（※）
```

(※)はジョブスクリプトに記載してバッチ形式で実行します。

例 2) OpenMP 並列による 8 スレッド実行

```
$ module load BaseVEC/2025
$ nfort -fopenmp test.f             # OpenMP を有効にしてコンパイル
$ export OMP_NUM_THREADS=8          # 実行スレッド数を指定（※）
$ ./a.out                           # 実行ファイルを実行（※）
```

(※)はジョブスクリプトに記載してバッチ形式で実行します。

4.4.3. MPI 並列実行を行う場合のコンパイル方法

Vector Engine 向けの MPI 実行では NEC MPI を使用します。コンパイルコマンドは以下になります。フロントエンド上でコンパイルしてください。

コンパイラのオプション情報は"-help"オプションを指定してご確認ください。

プログラム言語	NEC MPI
Fortran	mpinfort
C	mpincc
C++	mpinc++

例) MPI 並列による 8 プロセス（4VE x 2 プロセス）実行

```
$ module load BaseVEC/2025
$ mpinfort mpi_test.f                   # コンパイル
$ mpirun -np 8 -ve 0-3 ./a.out          # mpirun による実行を指定(※1)
$ mpirun -np 8 -venode -nn 4 ./a.out    # mpirun による実行を指定(※2)
$ mpirun -np 8 -venode -node 0-3 ./a.out # mpirun による実行を指定(※3)
```

(※1)(※2)(※3)はジョブスクリプトに記載してバッチ形式で実行します。

4.4.4. ライブラリの利用

ベクトル計算環境向けプログラムにおけるライブラリの利用方法を記載します。

(1) NEC Numeric Library Collection (NEC NLC)

NEC NLC はベクトル計算環境の SX-Aurora TSUBASA システム用に最適化された数値演算ライブラリです。NEC SX-ACE システムで提供していた ASL、MathKeisan を包含し、BLAS、LAPACK、ScaLAPACK 等をはじめとした多様なライブラリの利用が可能です。

本システムでは、BaseVEC モジュールの環境内に次の 2 つのモジュールが整備されています。

・necnlc モジュール(32bit 整数版)

・necnlc-mpi モジュール(32bit 整数+分散メモリ並列)

以下に本システムでのライブラリのリンク例を記載します。

・分散メモリ並列なし

```
# ASL ネイティブインターフェース 32bit 整数版ライブラリを静的リンク
$ module load BaseVEC/2025
$ module load necnlc/3.1.0
$ nfort -c sample.f                          # コンパイル
$ nfort -lasl_sequential -static sample.o    # リンク
```

・分散メモリ並列あり

```
# ScaLAPACK 分散メモリ並列版ライブラリを静的リンク
$ module load BaseVEC/2025
$ module load necnlc-mpi/3.1.0
$ mpinfort -c sample.f                                  # コンパイル
$ mpinfort -lscalapack -llapack -lblas_sequential ¥     # リンク
                -static sample.o
```

※ -static オプションを付与せず動的リンクとすることも可能です。

NEC NLC の詳細な利用方法は、公式のオンラインドキュメントをご参照ください。

➢ NEC Numeric Library Collection 2.3.0 ユーザーズガイド
https://www.hpc.nec/documents/sdk/SDK_NLC/UsersGuide/main/ja/index.html

(2) HDF5

HDF5 は、大量データを階層的に構造化して格納することが可能なファイルを生成するためのライブラリで、計算環境毎に整備されています。ライブラリを利用するためには、コンパイル時に-lhdf5 のオプションを付与します。以下はライブラリのリンク例です。

```
$ module load BaseVEC
$ module load hdf5
$ ncc -o h5-sample h5-sample.c -lhdf5 -ldl
```

(3) NetCDF

NetCDF は、科学的な多次元データを格納する目的で、共通データ形式として広く使われているバイナリファイルフォーマットを利用できるライブラリです。本システムでは、計算環境毎に、C 言語/C++言語/FORTRAN 言語のそれぞれから利用できるように整備されています。ライブラリを利用するためには、コンパイル時に-lnetcdf 等 のオプションを付与します。

以下はライブラリのリンク例です。

・C 言語

```
$ module load BaseVEC
$ module load netcdf-c
$ ncc -o ncdf-sample ncdf-sample.c -ldl -lnetcdf -lhdf5_hl -lhdf5 -lsz
```

・C++言語

```
$ module load BaseVEC
$ module load netcdf-cxx
$ nc++ -o ncdf-sample ncdf-sample.cpp ¥
        -lnetcdf_c++4 -lnetcdf -lhdf5 -lhdf5_hl -ldl -lsz
```

・FORTRAN 言語

```
$ module load BaseVEC
$ module load netcdf-fortran
$ nfort -o ncdf-sample ncdf-sample.f90 -lnetcdff -lnetcdf -lhdf5 -lhdf5_hl -ldl -lsz
```

(4) PnetCDF

PnetCDF は、NetCDF 形式のファイルに対して、並列プログラムから同時アクセスを行うためのライブラリです。本システムでは、計算環境毎に、C 言語/C++言語/FORTRAN 言語のそれぞれから利用できるように整備されています。ライブラリを利用するためには、コンパイル時に-lpnetcdf 等 のオプションを付与します。

以下はライブラリのリンク例です。

・C 言語

```
$ module load BaseVEC
$ module load pnetcdf-c
$ mpincc -o pncdf-sample pncdf-sample.c -lpnetcdf -lmpi
```

・C++言語

```
$ module load BaseVEC
$ module load pnetcdf-cxx
$ mpinc++ -o pncdf-sample pncdf-sample.cpp -lpnetcdf -lmpi -lmpi++
```

・FORTRAN 言語

```
$ module load BaseVEC
$ module load netcdf-fortran
$ mpinfort -o pncdf-sample pncdf-sample.f90 -lpnetcdf -lmpi
```

4.5. GPGPU 計算環境向けプログラムのコンパイル

GPGPU 計算環境向けプログラムのコンパイル方法を記載します。ここでは、推奨環境である BaseGPU 上でのコンパイル方法を記載します。プログラムコンパイル時には、BaseGPU 環境を事前に読み込んでください。

```
$ module load BaseGPU/2025
```

4.5.1. シリアル実行

コンパイルは、フロントエンド上で実施します。

各プログラム言語のコンパイルコマンドは以下になります。

プログラム言語	NVidia HPC SDK
FORTRAN	nvfortran
C	nvc
C++	nvc++
CUDA	nvcc

以下はコンパイル例です。”-o”オプションにより、作成する実行ファイル名を指定できます。

指定しなかった場合、ファイル名が a.out で作成されます。

最初に module コマンドで「NVidia HPC SDK」の環境変数設定モジュールをロードする必要があります。BaseGPU モジュールをロードすると、自動的に標準バージョンのコンパイラがロードされます。

```
$ module load BaseGPU/2025

$ nvfortran -o sample sample.f90
$ nvc -o sample sample.c
$ nvc++ -o sample sample.cpp
```

4.5.2. スレッド並列実行

スレッド並列で実行する場合は、コンパイル時に以下のオプションを指定して下さい。

並列モデル	NVidia HPC SDK
OpenMP	-mp
自動並列	-Mconcur

実行スレッド数は、実行時に環境変数 OMP_NUM_THREADS に<実行スレッド数>を指定して指定義します。

以下はコンパイル例です。最初に module コマンドで「NVidia HPC SDK」の環境変数設定モジュールをロードする必要があります。

```
$ module load BaseGPU/2025

$ nvfortran -o sample_omp -mp sample_omp.f90
$ nvfortran -o sample_smp -Mconcur sample_smp.f90
```

4.5.3. MPI 並列実行

MPI 環境は OpenMPI をご利用いただけます。

コンパイルコマンドは以下の通りです。

プログラム言語	OpenMPI の場合
FORTRAN	Mpifort
C	Mpicc
C++	mpic++

以下はコンパイル例です。

```
$ module load BaseGPU/2025

$ mpifort -o sample_mpi sample_mpi.f90
$ mpicc   -o sample_mpi sample_mpi.c
$ mpic++ -o sample_mpi sample_mpi.cpp
```

4.5.4. CUDA 用コンパイル

SQUID の GPU ノード群では、GPGPU を利用した数値計算が可能です。本ページでは CUDA を用いた自作プログラムをコンパイルする手順について記載しています。

GPU 向けの開発環境として、NVidia HPC SDK では「NVCC Compiler」が利用可能です。「NVCC Compiler」は、CUDA Fortran 用コンパイラ(nvfortran)と CUDA C 言語用コンパイラ(nvcc)から構成されています。CUDA Fortran の場合はプログラム拡張子が".cuf"または".CUF"、CUDA C の場合はプログラム拡張子が".cu"または".CU"になります。

プログラム言語	NVidia HPC SDK
CUDA Fortran	nvfortran
CUDA C	nvcc

以下はコンパイル例です。

- CUDA Fortran コンパイラ

```
$ module load BaseGPU/2025

$ nvfortran -o sample_cuda sample_cuda.cuf
```

- CUDA C コンパイラ

```
$ module load BaseGPU/2025

$ nvcc -o sample_cuda sample_cuda.cu
```

4.5.5. OpenACC 用オプション

OpenACC を使用したプログラムをコンパイルする際は以下のオプションを指定してください。

プログラム言語	NVidia HPC SDK
OpenACC C	-acc

以下はコンパイル例です。

```
$ module load BaseGPU/2025

$ nvc -o sample_acc -acc sample_openacc.c
```

4.5.6. ライブラリの利用

GPGPU 計算環境向けプログラムにおけるライブラリの利用方法を記載します。

ライブラリ利用に関して、GPGPU 計算環境向け FORTRAN コンパイラ(nvfortran)に対して注意事項があります。nvfortran には、環境変数に従って include ファイルの検索パスを変更する機能がありません。このため、module コマンドで設定された CPATH 環境変数を、コンパイル時に -I オプションにて引き渡す必要があります。実行例は以下となります。

```
$ nvfortran -o sample sample.f90   -I${CPATH}
```

(1) HDF5

HDF5 は、大量データを階層的に構造化して格納することが可能なファイルを生成するためのライブラリです。計算環境毎に整備されています。ライブラリを利用するためには、コンパイル時に -lhdf5 のオプションを付与します。

以下はライブラリのリンク例です。

```
$ module load BaseGPU
$ module load hdf5
$ nvc -o h5-sample h5-sample.c -lhdf5 -R${LD_RUN_PATH}
```

※ OS 標準の hdf5 パッケージと競合するケースがあるため、-R${LD_RUN_PATH} オプションを付加し、該当パッケージを優先設定します。

(2) NetCDF

NetCDF は、科学的な多次元データを格納する目的で、共通データ形式として広く使われているバイナリファイルフォーマットを利用できるライブラリです。本システムでは、計算環境毎に、C 言語/C++言語/FORTRAN 言語のそれぞれから利用できるように整備されています。ライブラリを

利用するためには、コンパイル時に-lnetcdf 等 のオプションを付与します。

以下はライブラリのリンク例です。

・C 言語

```
$ module load BaseGPU
$ module load netcdf-c
$ nvc -o ncdf-sample ncdf-sample.c -lnetcdf
```

・C++言語

```
$ module load BaseGPU
$ module load netcdf-cxx
$ nvc++ -o ncdf-sample ncdf-sample.cpp -lnetcdf_c++4   -lnetcdf
```

・FORTRAN 言語

```
$ module load BaseGPU
$ module load netcdf-fortran
$ nvfortran -o ncdf-sample ncdf-sample.f90   -I${CPATH} -lnetcdff -lnetcdf
```

(3) PnetCDF

PnetCDF は、NetCDF 形式のファイルに対して、並列プログラムから同時アクセスを行うためのライブラリです。本システムでは、計算環境毎に、C 言語/C++言語/FORTRAN 言語のそれぞれから利用できるように整備されています。ライブラリを利用するためには、コンパイル時に-lpnetcdf 等 のオプションを付与します。

以下はライブラリのリンク例です。

・C 言語

```
$ module load BaseGPU
$ module load pnetcdf-c
$ mpicc -o pncdf-sample pncdf-sample.c -lpnetcdf
```

・C++言語

```
$ module load BaseGPU
$ module load pnetcdf-cxx
$ mpic++ -o pncdf-sample pncdf-sample.cpp -lpnetcdf
```

・FORTRAN 言語

```
$ module load BaseGPU
$ module load pnetcdf-fortran
$ mpifort -o pncdf-sample pncdf-sample.f90 -I${CPATH} -lpnetcdf
```

4.6. GNU Compiler Collection によるコンパイル

本システムでは、GNU Compiler Collection を利用することが可能です。ここでは、ベース環境である BaseGCC 上でのコンパイル方法を記載します。プログラムコンパイル時には、BaseGCC 環境を事前に読み込んでください。

```
$ module load BaseGCC/2025
```

4.6.1. シリアル実行

コンパイルは、フロントエンド上で実施します。

各プログラム言語のコンパイルコマンドは以下になります。

プログラム言語	GNU Compiler Collection
FORTRAN	gfortran
C	gcc
C++	g++

以下はコンパイル例です。

”-o”オプションにより、作成する実行ファイル名を指定できます。指定しなかった場合、ファイル名が a.out で作成されます。

最初に module コマンドで GNU コンパイラの環境変数設定モジュールをロードする必要があります。BaseGCC モジュールをロードすると自動的に、標準バージョンの GNU コンパイラがロードされます。

```
$ module load BaseGCC/2025

$ gfortran -o sample sample.f90
$ gcc -o sample sample.c
$ g++ -o sample sample.cpp
```

4.6.2. スレッド並列実行

スレッド並列で実行する場合は、コンパイル時に以下のオプションを指定して下さい。

並列モデル	GNU Compiler Collection
OpenMP	-fopenmp

実行スレッド数は、実行時に環境変数 OMP_NUM_THREADS に<実行スレッド数>を指定して定義します。

以下はコンパイル例です。最初に module コマンドで GNU コンパイラの環境変数設定モジュールをロードする必要があります。

```
$ module load BaseGCC/2025

$ gfortran -o sample_omp -fopenmp sample_omp.f90
```

4.7. コンテナの利用方法

本システムでは、Singularity を用いた、コンテナによるプログラム実行環境を整えることが可能です。本項では、コンテナイメージの準備、およびコンテナのカスタマイズとビルドの方法について説明します。

コンテナの利用方法の概要は、「1.5.4 コンテナ利用」を参照ください。

4.7.1. コンテナイメージの準備

コンテナイメージは、SQUID 内のローカルレジストリや、インターネット上の公開レジストリから取得することが可能です。また、ご自身で作成したコンテナイメージを SQUID 内にアップロードすることもできます。

- **ローカルのワークステーションからシステム内に転送する**

イメージファイルとして準備されたコンテナイメージは、通常のファイルと同じように scp コマンド等で転送が可能です。本システム内にファイルを転送する方法については、「2.3 ファイル転送方法」を参照ください。

- **SQUID ローカルレジストリからイメージを取得する**

SQUID のローカルレジストリで公開しているコンテナイメージを取得する手順を説明します。例として、ローカルレジストリ/master_image に登録した test というコンテナイメージを取得する場合には、以下のコマンドを実行します。

```
$ singularity build test.sif ¥
            oras://cntm:5000/master_image/test:1.0
```

コマンドの書式は以下の通りです。

```
$ singularity build <イメージファイル名> ¥
```

```
        oras://cntm:5000/<コンテナイメージパス>:<tag 名>
```

コンテナイメージの取得に成功すると、カレントディレクトリにコンテナイメージ（上記の例では test.sif）が作成されます。

➢ Singularity Library からイメージを取得する

singularity library から singularity コンテナイメージを取得する手順を説明します。

例として、singularity library より、centos のコンテナイメージを取得する場合には、以下のコマンドを実行します。

```
$ singularity build centos.sif ¥
        library://emmeff/centos/centos:8
```

コマンドの書式は以下の通りです。

```
$ singularity build <イメージファイル名> ¥
        library://<コンテナイメージパス>:<tag 名>
```

コンテナイメージの取得に成功すると、カレントディレクトリにコンテナイメージ（上記の例では centos.sif）が作成されます。

➢ Docker Hub からイメージを取得する

Docker Hub から Docker コンテナイメージを取得し、singularity のコンテナイメージとして保存する手順を説明します。例として、Docker Hub より、centos のコンテナイメージを取得する場合には以下のコマンドを実行します。

```
$ singularity build centos.sif  ¥
        docker://docker.io/library/centos:latest
```

コマンドの書式は以下の通りです。

```
$ singularity build <イメージファイル名> ¥
        docker://docker.io/<コンテナイメージパス>:<tag 名>
```

コンテナイメージの取得に成功すると、カレントディレクトリにコンテナイメージ（上記の例では centos.sif）が作成されます。

➢ 参考：NVIDIA GPU CLOUD(NGC)からイメージを取得する

NVIDIA GPU CLOUD は NVidia 社が提供している WEB サイトで、GPGPU を利用する上でのドキュメントや、各種ユーティリティが公開されています。Docker コンテナの形式で、コンテナイメージも公開されており、本システム上から取得が可能です。

※取得には、利用者自身で NGC のユーザ登録をしていただく必要があります。

- NGC Catalog Containers
  https://ngc.nvidia.com/catalog/containers

以下の例では、NGC 上でのユーザ登録ならびに docker コンテナ用の API Key の発行が完了している前提で手順を説明します。

まず、環境変数に NGC ログイン用のユーザ名と API Key を設定します。ユーザ名は、$outhtoken 固定で、API Key は NGC サイト内で生成した Key を入力します。

```
$ export SINGULARITY_DOCKER_USERNAME='$oauthtoken'
$ export SINGULARITY_DOCKER_PASSWORD=<API Key>
```

次にコンテナイメージを取得します。例として、NGC より HPC-Benchmarks 21.4-hpl を取得する場合には、以下のコマンドを実行します。

```
$ singularity build hpc-benchmarks:21.4-hpl.sif ¥
        docker://nvcr.io/nvidia/hpc-benchmarks:21.4-hpl
```

コマンドの書式は以下の通りです。

```
$ singularity build <イメージファイル名> ¥
        docker://nvcr.io/nvidia/<コンテナイメージパス>:<tag 名>
```

コンテナイメージの取得に成功すると、カレントディレクトリにコンテナイメージ（上記の例では hpc-benchmarks:21.4-hpl.sif ）が作成されます。

4.7.2. コンテナイメージのカスタマイズとビルド

コンテナイメージのカスタマイズ、およびビルドの方法について説明します。

SQUID 内でコンテナイメージをカスタマイズしビルドするには、sandbox 経由でカスタマイズしビルドする方法と、def ファイルにカスタマイズ内容を記載しビルドする方法の二種類があります。

- **sandbox 経由でカスタマイズする方法**

sandbox を経由してカスタマイズし、ビルドする方法について説明します。

(1) sandbox の作成

最初に、コンテナイメージから sandbox を作成します。sandbox を作成したいディレクトリに移動し、sandbox 作成用のコマンドを実行します。この時、移動するディレクトリのグループ所有権とプライマリグループが同じである必要があるため、ホーム領域と、拡張領域や高速領域では、作成するコマンドが異なります。また、-f(fakeroot)オプションを付けた場合、作成したユーザでは sandbox を完全には削除できないことがあります。この場合は（４）sandbox の削除に記載した手順に従って、sandbox を削除してください。

例として、mySandbox というディレクトリに test.sif というイメージファイルがある際に、test という sandbox を作成する場合のコマンド実行例を記載します。

・ホーム領域で sandbox を作成する場合

```
$ cd ~/mySandbox
$ singularity build -f --sandbox --fix-perms test test.sif
```

・拡張領域・高速領域で sandbox を作成する場合

```
$ cd /sqfs/work/<グループ名>/<利用者番号>/mySandbox
$ newgrp <グループ名>
$ singularity build -f --sandbox --fix-perms test test.sif
```

singularity コマンドの書式は以下の通りです。

```
$ singularity build -f --sandbox --fix-perms <sandbox 名> <イメージファイル名>
```

(2) コンテナの起動

続いて、sandbox をコンテナとして起動します。

```
$ singularity run -f -w test
```

コマンドの書式は以下の通りです。

```
$ singularity run -f -w <sandbox 名>
```

コンテナの起動に成功すると、以下に示す Singularity のプロンプトが表示されます。

```
Singularity>
```

上記のプロンプトより、dnf、pip によるパッケージの追加などを実施します。パッケージの操作に使用するコマンドは、コンテナに格納した OS のディストリビューションにより異なります。

コンテナイメージのカスタマイズ完了後、以下のコマンドでコンテナを停止します。

```
Singularity> exit
```

(3) イメージファイル化

最後に、コンテナイメージのビルドを行い、sif ファイルを生成します。

```
$ singularity build -f test.sif test
```

コマンドの書式は以下の通りです。

```
$ singularity build -f <sif ファイル名> <sandbox 名>
```

ビルドに成功すると、カレントディレクトリに sif ファイルが作成されます。

(4) sandbox の削除

通常、sandbox は rm コマンドで削除することが可能です。

例として、mySandbox というディレクトリにある test という sandbox を削除する場合、以下のコマンドを実行します。

```
$ cd ~/mySandbox
$ rm -fr test
```

コマンドの書式は以下の通りです。

```
$ rm -fr <sandbox 名>
```

ただし、使用するコンテナイメージにより、sandbox を作成したユーザでは、sandbox を完全には削除できないことがあります。この場合は任意のコンテナを特権で起動し、コンテナ内から sandbox を削除することが可能です。

例として、mySandbox というディレクトリにある test という sandbox を、alpine のコンテナイメージを利用して削除する場合、以下のコマンドを実行します。

```
$ singularity exec -f --bind ~/mySandbox library://alpine rm -r test
```

コマンドの書式は以下の通りです。

```
$ singularity exec -f --bind <sandbox が存在するパス> <任意のコンテナイメージパス> ¥
 rm -r <sandbox 名>
```

➢ def ファイルにカスタマイズ内容を記載する方法

def ファイルにカスタマイズ内容を記載し、ビルド時にカスタマイズする方法について説明します。

(1) def ファイルの作成

まずは、ビルドに使用する def ファイルを作成します。

例として、ローカルレジストリに登録された test をベースコンテナイメージとしてカスタマイズを行う場合、def ファイルは以下のようになります。

```
1   Bootstrap: oras
2   From: cntm:5000/master_image/test:1.0
3
4   %files
5            ./test.conf /opt/test.conf
6            ./test_start.sh /opt/test_start.sh
7
8   %post
9           dnf install -y net-tools
10          chmod 755 /opt/test_start.sh
11
12  %runscript
13          /opt/test_start.sh
```

- def ファイルの概要

  (ア)　Bootstrap、From

  ベースイメージの種類、場所を記載します。

  (イ)　%file

  ホスト OS からコンテナにコピーしたいファイルを記載します。

  (ウ)　%post

  カスタマイズ用のコマンドを記載します。

  (エ)　%runscript

  コンテナ起動時に自動実行する処理を記載します。

※def ファイルの詳細については、singularity のマニュアルをご参照ください。

https://sylabs.io/guides/3.7/user-guide/definition_files.html

(2) イメージのビルド

def ファイルの作成完了後、コンテナイメージのビルドを行い、sif ファイルを生成します。

例として、test.def という def ファイルを使用してビルドを行い、test.sif を作成するには、以下のコマンドを実行します。

```
$ singularity build -f test.sif test.def
```

コマンドの書式は以下の通りです。

```
$ singularity build -f <sif ファイル名> <def ファイル名>
```

ビルドに成功すると、カレントディレクトリに sif ファイルが作成されます。

4.8. Python の利用方法

本システムでは、Python 言語環境として、Python3 及び Python2 を利用できます。Python 言語利用時には、ベース環境である BasePy を読み込んで利用します。

```
$ module load BasePy/2025
```

なお、BasePy を読み込んだ場合の標準 Python 環境は、Python3 になります。Python2 利用時には、Python2 の追加モジュールを読み込みます。

```
$ module load BasePy/2025
$ module load python3/3.11
```

また、GPU 対応の Python3 利用時には、モジュールを切り替えます。

```
$ module load BasePy/2025
$ module --force switch python3/3.6 python3/3.6.GPU
$ module load BaseGPU/2025
$ module load cudnn/8.2.0.53
```

4.8.1. 対話モードでの利用

Python 言語は、CLI 上で Python 言語を入力しながら対話的に実行結果を確認する対話モードの利用が可能です。本システム上での対話モードの起動、実行、終了の例は以下の通りです。

(1) Python3

```
# モジュールの読み込み
$ module load BasePy

# 対話モードの起動
$ python3
[GCC Intel(R) C++ gcc 8.3.1 mode] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>

# Python 言語の実行
>>> print("hellow python")          # 実行内容
hello python                        # 実行結果

# 対話モードの終了
>>> exit()
$
```

(2) Python2

```
# モジュールの読み込み
$ module load BasePy
$ module load python2

# 対話モードの起動
$ python2
Python 2.7.18 (default, Apr 19 2021, 13:14:04)
[GCC Intel(R) C++ gcc 8.3.1 mode] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>

# Python 言語の実行
```

```
>>> print "hello python"          # 実行内容
hello python                      # 実行結果

# 対話モードの終了
>>> exit()
$
```

4.8.2. プログラム(スクリプト)実行

通常のプログラミング言語のように、テキストエディタ等で作成した Python プログラム(スクリプト)を一括実行することも可能です。本システム上での Python プログラムの実行手順例を以下に示します。

(1) Python3

```
# モジュールの読み込み
$ module load BasePy

# プログラム実行
$ python3   sample.py
```

(2) Python2

```
# モジュールの読み込み
$ module load BasePy
$ module load python2

# プログラム実行
$ python2 sample.py
```

4.8.3. Python モジュールの追加

Python 言語は、PyPI(Python Package Index)において、多数の Python モジュールが公開されており、必要なものは利用者自身の環境に追加インストールすることが可能です。

- PyPI
  https://pypi.org/

モジュールの追加インストールは、pip コマンド(pip3/pip2)を利用します。pip コマンドの主要オ

プションは以下の通りです。

コマンド	説明
pipX list	インストール済み Python モジュール一覧の表示
pipX install [pymod]	Python モジュールのインストール
pipX show [pymod]	Python モジュールの詳細表示
pipX uninstall [pymod]	Python モジュールのアンインストール

※ pipX の部分は、Python3 の場合は pip3 、Python2 の場合は、pip2 となります。

本システムで標準整備している Python モジュールに加えて、利用者自身で追加インストールする際の実行例は以下の通りです。例では dumper モジュールを追加インストールしています。

※公開済みの本システムで標準整備した Python モジュールに対して、システム側で追加インストールは行いませんので、必要なモジュールは利用者自身で追加インストールしていただくよう、お願いします。

(1) Python3

```
# モジュールの読み込み
$ module load BasePy

# モジュールの追加インストール
$ pip3 install dumper
```

(2) Python2

```
# モジュールの読み込み
$ module load BasePy
$ module load python2

# モジュールの追加インストール
$ pip2 install dumper
```

4.8.4. ライブラリの利用

Python 言語では、TesorFlow、Keras、Pytorch ライブラリを使用することが可能です。

各サンプルプログラムは以下のとおりです。

(1) TensorFlow(sample_tensorflow.py)

```
import tensorflow as tf
```

```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
predictions = model(x_train[:1]).numpy()
predictions

tf.nn.softmax(predictions).numpy()
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
loss_fn(y_train[:1], predictions).numpy()
# 2.835742

model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test,  y_test, verbose=2)

probability_model = tf.keras.Sequential([
  model,
  tf.keras.layers.Softmax()
])
probability_model(x_test[:5])
```

(2) Keras(sample_keras.py)

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Model / data parameters
```

```
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()

batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

(3) Pytorch(sample_pytorch.py)

```
import torch
import math

dtype = torch.float
device = torch.device("cpu")
# device = torch.device("cuda:0") # Uncomment this to run on GPU

# Create random input and output data
x = torch.linspace(-math.pi, math.pi, 2000, device=device, dtype=dtype)
y = torch.sin(x)

# Randomly initialize weights
a = torch.randn((), device=device, dtype=dtype)
b = torch.randn((), device=device, dtype=dtype)
c = torch.randn((), device=device, dtype=dtype)
d = torch.randn((), device=device, dtype=dtype)

learning_rate = 1e-6
for t in range(2000):
    # Forward pass: compute predicted y
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

    # Compute and print loss
    loss = (y_pred - y).pow(2).sum().item()
    if t % 100 == 99:
        print(t, loss)

    # Backprop to compute gradients of a, b, c, d with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_a = grad_y_pred.sum()
    grad_b = (grad_y_pred * x).sum()
    grad_c = (grad_y_pred * x ** 2).sum()
    grad_d = (grad_y_pred * x ** 3).sum()

    # Update weights using gradient descent
    a -= learning_rate * grad_a
```

```
    b -= learning_rate * grad_b
    c -= learning_rate * grad_c
    d -= learning_rate * grad_d

print(f'Result: y = {a.item()} + {b.item()} x + {c.item()} x^2 + {d.item()} x^3')
```

実行方法は以下のとおりです。

```
# モジュールの読み込み
$ module load BasePy

# TensorFlow の実行
$ python sample_tensorflow.py

# Keras の実行
$ python sample_keras.py

# Pytorch の実行
$ python sample_pytorch.py
```

5. プログラム実行方法

5.1. 共通事項

5.1.1. ジョブ管理システムの解説

SQUID ではジョブ管理システム「NEC NQSV」により、計算リソースのバッチ利用および会話的利用が可能です。フロントエンドからジョブ管理システムにジョブ実行を要求（ジョブを投入）します。要求されたジョブ要求は、他のジョブ要求の優先度や要求資源量、SQUID の利用状況などがジョブ管理システムによって加味・判断され、実行順序が決定されます。

SQUID では、多人数のユーザで共有利用されることが前提となっているため、このようなバッチ処理方式を採用しています。

※ NQSV の”リクエスト”と”ジョブ”

ジョブ管理システム「NEC NQSV」の概念として、製品マニュアルでは以下の用語が使われます。

・リクエスト：ユーザがフロントエンドから要求するバッチ利用の単位

・ジョブ：個々の計算ノード上で実行される、リクエスト内の実行単位

ただし、本ドキュメント上は、他システムも含めた理解のしやすさを考慮し、"ジョブ"という用語を、フロントエンドから要求するバッチ利用の単位(NQSV のリクエストに相当)の意味で使用します。

5.1.2. 会話ジョブ投入方法

会話ジョブは、会話的（インタラクティブ）に計算ノードを利用するジョブです。会話ジョブを使用する場合は qlogin コマンドを使用します。

会話ジョブ投入コマンドのサンプル(汎用 CPU 計算環境で実行する例)を以下に記載します。

```
$ qlogin -q INTC --group=G01234 [オプション]
          ↑会話キュー名
                ↑課金対象にするグループ名
```

qlogin コマンドを実行すると、リクエスト ID が採番され、次のように標準出力に表示されます。

Request **1310.sqd** submitted to queue: INTC.

Waiting for **1310.sqd** to start.

5.1.3. バッチジョブ投入方法

フロントエンドからバッチジョブを投入するためには、以下の手順を実施します。

① ジョブスクリプトファイルの作成

② ジョブ管理システムへのバッチジョブ要求

(1) ジョブスクリプトファイルの作成

ジョブスクリプトファイルは、ユーザからのジョブ要求を記述するために必要となるファイルです。ユーザは、このジョブスクリプト内に、計算を行うための実行コマンドを書くとともに、ジョブ管理システムに対して要求するリソース量などをオプション(#PBS)として記述していきます。

ジョブスクリプトのサンプル(汎用 CPU 計算環境でシリアル実行する例)を以下に記載します。

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   # ↑ バッチリクエストを投入するキュー名の指定
5   #PBS --group=G01234
6   # ↑ 課金対象にするグループ名
7   #PBS -l elapstim_req=01:00:00
8   # ↑ ジョブの最大実行時間の要求値　1 時間の例
9   #PBS -l cpunum_job=76
10  # ↑ 使用する CPU コア数の要求値
11  #PBS -m b
12  # ↑ バッチリクエスト実行開始時にメールを送信
13  #PBS -M user@hpc.cmc.osaka-u.ac.jp
14  # ↑ 送信先アドレス
15
16  #------- Program execution -----------
17  module load BaseCPU/2025
18  # ↑ ベース環境をロードします
19
20  cd $PBS_O_WORKDIR
21  # ↑ qsub 実行時のカレントディレクトリへ移動
22  ./a.out
23  # ↑ プログラムの実行
```

(2) ジョブ管理システムへのバッチジョブ要求

スケジューラが提供するコマンド qsub を用いて以下のように行います。

```
$ qsub　[オプション]　[ジョブスクリプトファイル名]
```

qsub コマンドを実行すると、リクエスト ID が採番され、次のように標準出力に表示されます。

Request **1182.sqd** submitted to queue: SQUID.

qsub コマンドの主なオプションは以下の通りです。システムで共通しているオプション、ベクトルノード向けオプション、CPU ノード向けオプション、GPU ノード向けオプションがあります。利用するシステムに応じたオプションをご確認ください。

【共通オプション】

オプション	機能
-q [バッチキュー名]	バッチジョブを投入するキューを指定します。**(必須)**
--group=[グループ名]	指定グループでジョブを実行します。指定グループの予算が消費されます。**(必須)**
-l elapstim_req=[経過時間制限値]	バッチジョブの経過時間制限値を指定します。 指定がなければ各キューの既定値が適用されます。 時間制限を指定する際の形式は次の通りです。 hh:mm:ss　hh 時間 mm 分 ss 秒
-N [ジョブ名]	ジョブの名前を指定します。 指定がなければ、バッチスクリプト名がジョブ名になります。
-o [標準出力ファイル名]	バッチジョブの標準出力の出力ファイル名を指定します。 指定がなければ、ジョブ投入時のディレクトリに「ジョブ名.o リクエスト ID」のファイル名で出力されます。
-e [標準エラー出力ファイル名]	バッチジョブの標準エラー出力の出力ファイル名を指定します。 指定がなければ、リクエスト投入時のディレクトリに「ジョブ名.e リクエスト ID」のファイル名で出力されます。
-j [o,e]	バッチジョブの標準出力と標準エラー出力をマージします。 o：マージした結果を標準出力に出力します。 e：マージした結果を標準エラー出力に出力します。 （-j と o もしくは e の間にはスペースが必要です）
-M [メールアドレス]	メールの送信先を指定します。複数指定する場合は -M メールアドレス 1,メールアドレス 2 のように「,」で区切ってください。
-m [b,e,a]	バッチジョブの状態の変化についてのメールを送ります。 b：ジョブが開始したときにメールを送信 e：ジョブが終了したときにメールを送信 a：ジョブが異常終了したときにメールを送信

	（-m と b や e の間にはスペースが必要です） 複数を指定することができます。 例）開始時と終了時にメール通知 -m be
-v 環境変数	バッチジョブを実行するときに使用する環境変数を指定します。
-r { y | n }	バッチジョブのリラン可否を指定します。 y：リラン可能 n：リラン不可

【CPU ノード向けオプション】

オプション	機能
-b [ノード数]	ジョブを実行するノード数を指定します。
-T [使用 MPI ライブラリ名]	MPI 実行を行う場合に指定が必要です。 Intel コンパイラを利用している場合、IntelMPI が利用可能です。-T intmpi と指定してください。 NVIDIA HPC SDK コンパイラを利用している場合、OpenMPI が利用可能です。-T openmpi を指定してください。

【GPU ノード向けオプション】

オプション	機能
-l gpunum_job=[ノードあたりの GPU 数]	ノードあたりの利用する GPU 数を指定します。
--gpuum-lhost=[ノードあたりの GPU 数]	上記「-l gpunum_job」と同様です。
-T [使用 MPI ライブラリ名]	MPI 実行を行う場合に指定が必要です。 Intel コンパイラを利用している場合、IntelMPI が利用可能です。-T intmpi と指定してください。 NVIDIA HPC SDK コンパイラを利用している場合、OpenMPI が利用可能です。-T openmpi を指定してください。

【ベクトルノード向けオプション】

オプション	機能
--venode=[総 VE 数]	利用する総 VE 数を指定します。
--venum-lhost=[論理ホストあたりの VE 数]	論理ホストを構成する VE 数を指定します（大まかには、[1VH 内で確保する VE 数]とご理解ください）。
-T necmpi	SX-Aurora TSUBASA にて MPI 実行を行う場合に指定が必要です。

5.1.4. ジョブ管理システムのコマンドについて

投入したジョブの状態は、ジョブ管理システムの各種コマンドにて確認が可能です。

(1) バッチジョブの確認

投入したジョブの状況を確認する場合は qstat コマンドを使用します。

- ユーザ自身が投入したジョブの一覧を表示する場合

```
$ qstat
```

- 省略なく一覧を表示する場合

```
$ qstat  -l
```

上記はアルファベットの L の小文字です。

「--adjust-column」を同時に指定することで幅の最適化が行われます。

- 特定ジョブの詳細表示を行う場合

```
$ qstat  -f  [リクエスト ID]
```

また、ユーザが所属するグループが投入したジョブの状況を確認する場合は qstatgroup コマンドを使用します。

- ユーザが所属するグループが投入したジョブの一覧を表示する場合

```
$ qstatgroup
```

投入したリクエストの実行開始時刻を確認する場合は sstat コマンドを使用します。実行開始時刻が決まっていない場合、時刻は表示されません。

```
$ sstat
```

ジョブ実行中に、標準出力／標準エラー出力の内容を表示する場合は qcat コマンドを使用します。-e/-o を指定しなかった場合、ジョブスクリプトが表示されます。

```
$ qcat  -e  [リクエスト ID]   ※標準エラー出力の表示の場合
$ qcat  -o  [リクエスト ID]   ※標準出力の表示の場合
```

以下のオプションを組み合わせることも可能です。

-f　　ファイルの内容が増え続けるとき、追加されたデータを出力します。

-n　　指定した行数分表示します。（無指定時は 10 行分です。）

-b　　ファイルの先頭から表示します。（無指定時は最終行から表示されます。）

ジョブが終了すると、qstat コマンドでは表示されなくなります。

(2) バッチジョブの保留(ホールド)

投入したジョブを保留（ホールド）する場合は qhold コマンドを利用します。ホールドすることでスケジューリング対象から外れ、実行が開始されない状態となります。

```
$ qhold  [リクエスト ID]
```

(3) バッチジョブの保留解除(リリース)

保留解除（リリース）は qrls コマンドを使用します。ホールドを解除することにより、ジョブはホールド解除前の状態に戻り、再度スケジューリング対象となります。

```
$ qrls  [リクエスト ID]
```

(4) ジョブ情報の表示

ユーザ自身が過去に投入したジョブの情報を表示する場合は acstat コマンドを使用します。

- ユーザ自身が過去に投入したジョブの情報(コマンド実行時から過去 24 時間以内)を表示する場合

```
$ acstat
```

- ユーザ自身が過去に投入したジョブの情報(コマンド実行年度)を表示する場合

```
$ acstat -A
```

また、ユーザが所属するグループが過去に投入したジョブの情報を表示する場合は acstatgroup コマンドを使用します。

- ユーザが所属するグループが過去に投入したジョブの情報(コマンド実行時から過去 24 時間以内)を表示する場合

```
$ acstatgroup
```

- ユーザが所属するグループが過去に投入したジョブの情報(コマンド実行年度)を表示する場合

```
$ acstatgroup -A
```

(5) バッチジョブの削除

投入したリクエストを削除する場合は qdel コマンドを利用します。

```
$ qdel [リクエスト ID]
```

バッチジョブが実行状態(RUN)の場合は、最初に SIGTERM を送り、その後 SIGKILL が送られます。-g オプションで grace タイムの秒数を指定することにより、SIGKILL を送信するまでの待ち時間を指定することができます。指定がない場合は 5 秒です。

(6) キューの状態確認

バッチジョブが実行されるキューの状態を表示する場合は、qstat コマンドを使用します。

```
$ qstat -Q
```

5.1.5. NQSV における Core ファイル出力設定

NQSV を介して実行される会話ジョブおよびバッチジョブでは、NQSV 側で Core ファイルの出力制限（ulimit）が設定されます。

Core ファイルの出力サイズの既定値は全キュー共通で 0 に設定されているため、設定を行わない場合、Core ファイルは出力されません。実行されるプログラムにおいて、Core ファイルを出力したい場合は、ジョブスクリプト内で明示的に Core ファイル出力サイズを設定してください。

なお、Core ファイルの出力先は、プログラムを実行したディレクトリとなります。

オプション	機能
coresz_prc=[max_limit] 50mb を設定する場合 #PBS -l coresz_prc=50mb	プロセス単位でのコアファイルサイズ制限値 サイズの制限値は、以下の形式で指定してください。 [integer][.fraction][units] units に指定可能な単位は以下の通りです。 units を指定しない場合は、バイトと解釈されます b　：バイト kb　：キロバイト mb　：メガバイト

	gb　：ギガバイト tb　：テラバイト

Core ファイルの出力形式は以下のとおりです。

項目	設定値	備考
Core ファイル出力先	core.%P.%u.%g.%s.%t.%c.%h	%P：プロセス PID %u:ダンプされたプロセスの実ユーザ %g:ダンプされたプロセスの実グループ %：ダンプを引き起こしたシグナル番号 %t：ダンプ生成時刻 %c：core ファイルサイズ上限(Soft) %h：ホスト名

5.2. ジョブクラス

SQUID のジョブクラスについて記載します。それぞれのジョブクラスは、ジョブ管理システム上のキューに対応しており、利用者はキューにジョブを投入することで計算環境の利用が可能です。

キューは、利用者がジョブを直接投入する投入キューと、実行順を待ち合わせる実行キューに分かれます。

5.2.1. 汎用 CPU 計算環境向けジョブクラス

利用方法	ジョブクラス	利用可能経過時間	利用可能最大 Core 数	利用可能メモリ	同時利用可能ノード数	備考
共有利用	SQUID	24 時間	38,912core (76c/ノード)	124TiB (248GB/ノード)	512	ノード内は占有利用
	SQUID-R	24 時間	38,912core (76c/ノード)	124TiB (248GB/ノード)	512	※1
	SQUID-H	24 時間	38,912core (76c/ノード)	124TiB (248GB/ノード)	512	※2
	SQUID-S	24 時間	38core	124GiB	1	※3

	DBG	10 分	152core (76c/ノード)	496GiB (248GB/ノード)	2	デバッグ用
	INTC	10 分	152core (76c/ノード)	496GiB (248GB/ノード)	2	インタラクティブ利用
占有利用	mySQUID	無制限	76core × 占有ノード数	248GiB × 占有ノード数	占有数	

※1 NW 帯域が狭い経路の利用を許容して、実行待ち時間を短縮されたい方向け

※2 ポイント消費を多くして高優先度ジョブを投入し、実行待ち時間を短縮されたい方向け

※3 他のジョブとのノード内の共有を許容して、ポイント消費を抑えたい方向け

5.2.2. ベクトル計算環境向けジョブクラス

利用方法	ジョブクラス	利用可能経過時間	利用可能最大 Core 数	利用可能メモリ	同時利用可能 VE 数	備考
共有利用	SQUID	24 時間	2,560core (10c/VE)	12TiB (48GB/VE)	256	
	SQUID-H	24 時間	2,560core (10c/VE)	12TiB (48GB/VE)	256	※1
	SQUID-S	24 時間	40core	192GiB	4	※2
	DBG	10 分	40core (10c/VE)	192GiB (48GB/VE)	4	デバッグ用
	INTV	10 分	40core (10c/VE)	192GiB (48GB/VE)	4	インタラクティブ利用
占有利用	mySQUID	無制限	10core × 占有 VE 数	48GiB × 占有 VE 数	占有数	

※1 ポイント消費を多くして高優先度ジョブを投入し、実行待ち時間を短縮されたい方向け

※2 他のジョブとのノード内の共有を許容して、ポイント消費を抑えたい方向け

5.2.3. GPGPU 計算環境向けジョブクラス

利用方法	ジョブクラス	利用可能経過時間	利用可能最大 Core 数	利用可能メモリ	同時利用可能ノード数	備考
共有利用	SQUID	24 時間	2,432core (76c/ノード)	15.75TiB (504GB/ノード)	32	
	SQUID-H	24 時間	2,432core (76c/ノード)	15.75TiB (504GB/ノード)	32	※1
	SQUID-S	24 時間	38core	252GiB	1	※2

	DBG	10 分	152core (76c/ノード)	1,008GiB (504GB/VE)	2	デバッグ用
	INTG	10 分	152core (76c/ノード)	1,008GiB (504GB/VE)	2	インタラクティブ利用
占有利用	mySQUID	無制限	76core × 占有ノード数	504GiB × 占有ノード数	占有数	

※1 ポイント消費を多くして高優先度ジョブを投入し、実行待ち時間を短縮されたい方向け

※2 他のジョブとのノード内の共有を許容して、ポイント消費を抑えたい方向け

5.3. 汎用 CPU 計算環境の使い方

本項では汎用計算環境向けジョブスクリプトについて説明します。

5.3.1. シリアル実行利用方法

１ノード内でのプログラム実行を想定したジョブスクリプトのサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6
7   #------- Program execution -----------
8   module load BaseCPU/2025
9   module load xxx/xxx
10
11  cd $PBS_O_WORKDIR
12  ./a.out
```

- 9 行目

  コンパイル時に module load していたものを記載します。

このジョブスクリプトはプログラムの実行を行うだけのものになっていますが、オプションを追加で指定することで「ジョブ実行終了時にメールアドレスに通知を送信する」、「出力結果ファイルの名前を指定する」といった要求を行うことも可能です。

5.3.2. スレッド並列化利用方法

１ノード内でスレッド並列プログラム実行を想定したジョブスクリプトのサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）
- 総 CPU コア数　　：76（=38 コア×2CPU×1 ノード）

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
```

```
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -v OMP_NUM_THREADS=76
7
8   #------- Program execution -----------
9
10  module load BaseCPU/2025
11  module load xxx/xxx
12
13  cd $PBS_O_WORKDIR
14  ./a.out
```

- 6 行目

  並列実行数を指定します。

- 11 行目

  コンパイル時に module load していたものを記載します。

利用時の注意点

実行スクリプトにて、「OMP_NUM_THREADS」を忘れずに指定してください。

「OMP_NUM_THREADS」を指定しない場合、あるいは間違った値を指定してしまった場合、意図しない並列数での実行となる可能性があります。ご注意ください。

5.3.3. MPI 利用方法

4 ノードで 304 並列実行を想定したジョブスクリプトのサンプルです。

- キュー : SQUID
- グループ : G01234
- 経過時間 : 30 分（未指定の場合は 1 時間）
- 総 CPU コア数 : 304（=38 コア×2CPU×4 ノード）
- ノード数 : 4（1 ノード内実行時は指定不要）
- MPI ライブラリ : IntelMPI

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -b 4
7   #PBS -T intmpi
8
9   #------- Program execution -----------
```

```
10
11   module load BaseCPU/2025
12   module load xxx/xxx
13
14   cd $PBS_O_WORKDIR
15   mpirun ${NQSV_MPIOPTS} -np 304 ./a.out
```

- 6 行目

  ノード数を指定します。

- 7 行目

  MPI 利用時には -T intmpi を指定します。

- 12 行目

  コンパイル時に module load していたものを記載します。

- 15 行目

  mpirun の引数に${NQSV_MPIOPTS}を指定してください。この指定をとおして、ジョブが実行されるホストを MPI に渡すようになっています。

  環境変数 PBS_O_WORKDIR には qsub コマンドを実行したカレントディレクトリのパスが自動的に代入されています。

5.3.4. MPI＋ノード内並列 利用方法

4 ノードで 4 プロセス(ノードあたり 1 プロセス)を生成し、各プロセスは 76 スレッドを生成するプログラム実行を想定したジョブスクリプトのサンプルです。

- キュー : SQUID
- グループ : G01234
- 経過時間 : 30 分（未指定の場合は 1 時間）
- 総 CPU 数 : 304（=38 コア×2CPU×4 ノード）
- ノード数 : 4
- MPI ライブラリ : IntelMPI
- スレッド数 : 76（環境変数 OMP_NUM_THREADS で指定）

```
1    #!/bin/bash
2    #------- qsub option ------------
3    #PBS -q SQUID
```

```
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -b 4
7   #PBS -T intmpi
8   #PBS -v OMP_NUM_THREADS=76
9
10  #------- Program execution -----------
11
12  module load BaseCPU/2025
13  module load xxx/xxx
14
15  cd $PBS_O_WORKDIR
16  mpirun ${NQSV_MPIOPTS} -np 4 ./a.out
```

- 7 行目

  MPI 利用時には -T intmpi を指定します。

- 13 行目

  コンパイル時に module load していたものを記載します。

- 16 行目

  mpirun の引数に${NQSV_MPIOPTS}を指定してください。この指定をとおして、ジョブが実行されるホストを MPI に渡すようになっています。

  環境変数 PBS_O_WORKDIR には qsub コマンドを実行したカレントディレクトリのパスが自動的に代入されています。

5.3.5. 高度な使い方

➢ **Intel MPI における、ノード内プロセス数のマニュアル指定**

Intel MPI では、-ppn、-rr、-perhost オプション(I_MPI_PERHOST 環境変数含む)を指定することで、ノード内のプロセスを限定することが可能です。ただし、-machinefile オプションを併用した場合、-machinefile オプションが優先されます。

SQUID のジョブ管理システムでは、NQSV_MPIOPTS 環境変数の中に、-machinefile オプションを含んでいるため、-ppn、-rr、-perhost オプションは無効となります。ノード内のプロセス数を限定する場合は、-l cpunum_job オプションに必要コア数を指定することで可能です。ただし、コアのピニング等を併用するなどより細かい指定をする場合は、PBS_NODEFILE 環境変数を利用します。

以下は、PBS_NODEFILE 環境変数を利用して、2 ノードで 128 プロセス(ノードあたり 64 プロセス)を生成するジョブスクリプトのサンプルです。

- キュー　　: SQUID
- グループ　　: G01234
- 経過時間　　: 30 分（未指定の場合は 1 時間）
- 総プロセス数　　: 128（=64 プロセス×2 ノード）
- ノード数　　: 2（1 ノード内実行時は指定不要）
- MPI ライブラリ　　: IntelMPI

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -b 2
7   #PBS -T intmpi
8
9   #------- Program execution -----------
10
11  module load BaseCPU/2025
12  module load xxx/xxx
13
14  cd $PBS_O_WORKDIR
15  mpirun -hostfile ${PBS_NODEFILE} -np 128 -ppn 64 ./a.out
```

- 6 行目

  ノード数を指定します。

- 7 行目

  MPI 利用時には -T intmpi を指定します。

- 12 行目

  コンパイル時に module load していたものを記載します。

- 15 行目

  mpirun の引数に-hostfile ${PBS_NODEFILE} を指定してください。この指定をとおして、ジョブが実行されるホストを MPI が認識します。

  合わせて、-ppn 64 を指定することで、ノードあたり 64 プロセスの指定が可能です。

➢ VTune プロファイラによるパフォーマンス解析

VTune プロファイラを使用することで、ユーザーが開発したシリアル、およびマルチスレッド化

されたアプリケーションのパフォーマンス解析を行うことが可能です。

以下は、VTune プロファイラを使用して、パフォーマンス解析を行うジョブスクリプトのサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）
- 総プロセス数　　：2（=2 プロセス×1 ノード）
- MPI ライブラリ　　：IntelMPI

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group= G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -T intmpi
7
8   #------- Program execution -----------
9   module load BaseCPU/2025
10  module load xxx/xxx
11
12  cd $PBS_O_WORKDIR
13
14  aps --result-dir=./result --collection-mode=all mpirun ${NQSV_MPIOPTS} -ppn 2 -np
    2 ./a.out
```

- 6 行目

  MPI 利用時には -T intmpi を指定します。

- 9 行目

  コンパイル時に module load していたものを記載します。

- 14 行目

  aps コマンドとオプションに引き続き、mpirun コマンドを記載します。--result-dir には、解析結果の出力先パスを指定します。また、--collection-mode には、以下に示す収集したいデータのリストをカンマ区切りで指定します。この例では「すべて」を指定しています。

  hwc：ハードウェアカウンター

  omp：OpemMPI の統計

  mpi：MPI の統計

  all：すべて

バッチジョブを実行すると、カレントディレクトリに aps_report_YYYYMMDD_HHMISS.html の名前でレポートが出力され、--result-dir に指定したディレクトリに、パフォーマンス解析結果が出力されます。

なお、VTune プロファイラの使用方法、レポートの見方などにつきましては VTune プロファイラのユーザガイドをご参照ください。

5.4. ベクトル計算環境の使い方

本項ではベクトル計算環境向けジョブスクリプトについて説明します。

5.4.1. シリアル実行利用方法

1 VE 内でのプログラム実行を想定したジョブスクリプトのサンプルです。

- キュー : SQUID-S
- グループ : G01234
- 経過時間 : 30 分（未指定の場合は 1 時間）
- 総 VE 数 : 1

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID-S
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS --venode=1
7
8   #------- Program execution -----------
9   module load BaseVEC/2025
10  module load xxx/xxx
11
12  cd $PBS_O_WORKDIR
13  ./a.out
```

- 6 行目

  全体で 1VE 使用することを指定します。

- 10 行目

  コンパイル時に module load していたものを記載します。

5.4.2. スレッド並列化利用方法

１ノード内でスレッド並列プログラム実行を想定したジョブスクリプトのサンプルです。

- キュー　　：SQUID-S
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）
- スレッド数　　：10

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID-S
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -v OMP_NUM_THREADS=10
7   #PBS --venode=1
8
9   #------- Program execution -----------
10
11  module load BaseVEC/2025
12  module load xxx/xxx
13
14  cd $PBS_O_WORKDIR
    ./a.out
```

- 6 行目

  並列実行数を指定します。

- 12 行目

  コンパイル時に module load していたものを記載します。

利用時の注意点

実行スクリプトにて、「OMP_NUM_THREADS」を忘れずに指定してください。

「OMP_NUM_THREADS」を指定しない場合、あるいは間違った値を指定してしまった場合、意図しない並列数での実行となる可能性があります。ご注意ください。

5.4.3. MPI 利用方法

４０VE で 400 並列実行を想定したジョブスクリプトのサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）

- 総 VE 数　　：40
- MPI ライブラリ　　：NEC MPI
- 並列数　　：400(40VE×10 コア)

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS --venode=40
7   #PBS -T necmpi
8
9   #------- Program execution -----------
10
11  module load BaseVEC/2025
12  module load xxx/xxx
13
14  cd $PBS_O_WORKDIR
15  mpirun -venode -np 400 ./a.out
```

- 7 行目

  MPI 利用時には -T necmpi の指定が必須です。また necmpi 以外は利用できません。

- 12 行目

  コンパイル時に module load していたものを記載します。

- 15 行目

  MPI プロセスの VH および VE への割り当ては NQSV が自動的に行います。
  9VE 以上を使用する際にプロセス数が 8 の倍数でない場合、VE に割り当たるプロセス数が不均一になる場合があります。MPI プロセスを全て VE に割り当てる場合は、-venode オプションを指定することを推奨します。

5.4.4. MPI＋ノード内並列 利用方法

40VE で、40 プロセス（VE あたり 1 プロセス）を生成し、各プロセスは 10 スレッドを生成するプログラム実行を想定したジョブスクリプトのサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）
- 総 VE 数　　：40

- 論理ホストあたりの VE 数 ：8（1 ホスト当たり最大 8）
- MPI ライブラリ　　：NEC MPI
- スレッド数　　：10（環境変数 OMP_NUM_THREADS で指定）

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS --venode=40
7   #PBS -T necmpi
8   #PBS -v OMP_NUM_THREADS=10
9
10  #------- Program execution -----------
11
12  module load BaseVEC/2025
13  module load xxx/xxx
14
15  cd $PBS_O_WORKDIR
16  mpirun -venode -np 40 ./a.out
```

- 7 行目

  MPI 利用時には -T necmpi の指定が必須です。また necmpi 以外は利用できません。

- 8 行目

  スレッド数を指定します。VE あたりのスレッド数がコア数である、10 を超えないようにしてください。

- 13 行目

  コンパイル時に module load していたものを記載します。

- 16 行目

  MPI プロセスの VH および VE への割り当ては NQSV が自動的に行います。
  9VE 以上を使用する際にプロセス数が 8 の倍数でない場合、VE に割り当たるプロセス数が不均一になる場合があります。MPI プロセスを全て VE に割り当てる場合は、-venode オプションを指定することを推奨します。

5.5. GPGPU 計算環境の使い方

本項では GPGPU 計算環境向けジョブスクリプトについて説明します。

5.5.1. シリアル実行利用方法

1 ノード内でのプログラム実行を想定したジョブスクリプトのサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）
- GPU 数　　：1

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -l gpunum_job=1
7
8   #------- Program execution -----------
9   module load BaseGPU/2025
10  module load xxx/xxx
11
12  cd $PBS_O_WORKDIR
13  ./a.out
```

- 6 行目

  ノードあたり 8GPU 使用することを指定します。

- 10 行目

  コンパイル時に module load していたものを記載します。

5.5.2. スレッド並列化利用方法

1 ノード内でスレッド並列プログラム実行を想定したジョブスクリプトのサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）
- スレッド数　　：76

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -l gpunum_job=8
7   #PBS -v OMP_NUM_THREADS=76
8
```

```
9   #------- Program execution -----------
10
11  module load BaseGPU/2025
12  module load xxx/xxx
13
14  cd $PBS_O_WORKDIR
    ./a.out
```

- 7 行目

  並列実行数を指定します。

- 11 行目

  コンパイル時に module load していたものを記載します。

利用時の注意点

実行スクリプトにて、「OMP_NUM_THREADS」を忘れずに指定してください。

「OMP_NUM_THREADS」を指定しない場合、あるいは間違った値を指定してしまった場合、意図しない並列数での実行となる可能性があります。ご注意ください。

5.5.3. MPI 利用方法(OpenMPI)

2 ノードで合計 4 プロセスを生成する OpenMPI のプログラム実行を想定したジョブスクリプトのサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）
- ノード数　　：2
- MPI ライブラリ　：OpenMPI

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -b 2
7   #PBS -l gpunum_job=8
8   #PBS -T openmpi
9   #PBS -v NQSV_MPI_MODULE=BaseGPU/2025
10
11  #------- Program execution -----------
12
13  module load BaseGPU/2025
```

```
14   module load xxx/xxx
15
16   mpirun ${NQSV_MPIOPTS} -np 4 -npernode 2  ${PBS_O_WORKDIR}/mpi_prog
```

- 9 行目

  OpenMPI の起動時に必要となる環境変数をモジュール名で指定します。複数のモジュールを指定する場合は、コロン(:)区切りにて複数のモジュールを指定することが可能です。

  例) BaseGPU/2025 と cuda/12.6 を指定する場合

  -v NQSV_MPI_MODULE=BaseGPU/2025:cuda/12.6

- 13 行目

  module コマンドで MPI 環境設定を指定してください。これはマスターノードでの設定になります。スレーブノードへの設定は qsub オプション(スクリプトの 8,9 行目)で行います。

- 16 行目

  mpirun の引数に${NQSV_MPIOPTS}を指定してください。この指定をとおして、ジョブが実行されるホストを MPI に渡すようになっています。

  環境変数 PBS_O_WORKDIR には qsub コマンドを実行したカレントディレクトリのパスが自動的に代入されています。

5.5.4. MPI＋ノード内並列 利用方法(OpenMPI)

2 ノードで各ノード上に 2 個ずつプロセスを生成し、OpenMP により各プロセスで 12 個のスレッドを生成する OpenMPI のプログラム実行を想定したジョブスクリプトサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）
- ノード数　　：2
- MPI ライブラリ　：OpenMPI
- スレッド数　　：12（環境変数 OMP_NUM_THREADS で指定)

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -b 2
7   #PBS -l gpunum_job=8
8   #PBS -T openmpi
9   #PBS -v NQSV_MPI_MODULE=BaseGPU/2025
```

```
10  #PBS -v OMP_NUM_THREADS=12
11
12  #------- Program execution -----------
13
14  module load BaseGPU/2025
15  module load xxx/xxx
16
17  mpirun ${NQSV_MPIOPTS} -np 4 -npernode 2 --bind-to socket ¥
                                        --report-bindings  ${PBS_O_WORKDIR}/mpi_prog
```

- 9 行目

  OpenMPI の起動時に必要となる環境変数をモジュール名で指定します。複数のモジュールを指定する場合は、コロン(:)区切りにて複数のモジュールを指定することが可能です。

  例) BaseGPU/2025 と cuda/12.6 を指定する場合

  -v NQSV_MPI_MODULE=BaseGPU/2025:cuda/12.6

- 14 行目

  module コマンドで MPI 環境設定を指定してください。これはマスターノードでの設定になります。スレーブノードへの設定は qsub オプション(スクリプトの 8,9 行目)で行います。

- 17 行目

  mpirun の引数に${NQSV_MPIOPTS}を指定してください。この指定をとおして、ジョブが実行されるホストを MPI に渡すようになっています。

  環境変数 PBS_O_WORKDIR には qsub コマンドを実行したカレントディレクトリのパスが自動的に代入されています。

  OpenMPI において、1 プロセスを 1CPU に割り当て、12 コアに 12 スレッドを割り当てるためには、上記サンプルのように「--bind-to socket」等と設定してください。「--report-bindings」を指定することで、プロセスへのコアの割り当て状況を表示できます。

5.5.5. 高度な使い方

- **MPI プロセスが使用する GPU のマニュアル指定**

  SQUID のジョブ管理システムおよび OpenMPI では、各 MPI プロセスが使用する GPU について制御を行いません。このため、プログラムによっては MPI プロセス(ランク)間で使用 GPU がぶつかることがあります。

  MPI プロセス毎に使用する GPU を細かく制御したい場合は、プログラムを実行するためのラッパースクリプトを用意して、MPI プロセス(ランク)毎に使用する GPU 番号を指示することで対処

が可能です。

1 ノードで 8 個のプロセスを生成し、OpenMPI のプログラム実行を想定したジョブスクリプトサンプルです。ラッパースクリプトとして、CPU に集約配置する場合(mpiwrap-seq.sh)と CPU に分散配置する場合(mpiwrap-alt.sh)の 2 種を説明します。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 1 時間）
- ノード数　　：1
- MPI ライブラリ　　：OpenMPI

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -b 1
7   #PBS -l gpunum_job=8
8   #PBS -T openmpi
9   #PBS -v NQSV_MPI_MODULE=BaseGPU/2025
10
11  #------- Program execution -----------
12
13  module load BaseGPU/2025
14  WRAP=${PBS_O_WORKDIR}/mpiwrap-seq.sh
15
16  mpirun ${NQSV_MPIOPTS} -np 8 -npernode 8 ¥
17   --bind-to socket ${WRAP} ${PBS_O_WORKDIR}/mpi_prog
18
```

- 14 行目

  WRAP に使用するラッパースクリプトのパスを指定します。

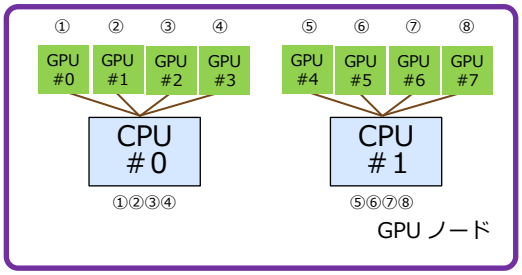
- 17 行目

  mpirun の引数に${WRAP}を指定してください。

  これにより、ラッパースクリプトを有効化することができます。

※ CPU に集約配置するラッパースクリプト例

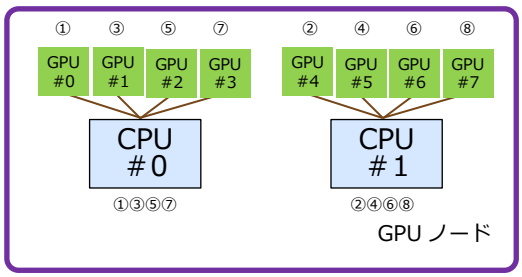
mpiwrap-seq.sh： ランク 0 から順に、図の順番で CPU、GPU を割当てます。




```
1   #!/bin/bash
2
3   echo LANK=${OMPI_COMM_WORLD_LOCAL_RANK}
4
5   case ${OMPI_COMM_WORLD_LOCAL_RANK} in
6       0 ) # CPU 0, GPU 0
7           export CUDA_VISIBLE_DEVICES=0
8           numactl -N 0 -l $@
9           ;;
10      1 ) # CPU 0, GPU 1
11          export CUDA_VISIBLE_DEVICES=1
12          numactl -N 0 -l $@
13          ;;
14      2 ) # CPU 0, GPU 2
15          export CUDA_VISIBLE_DEVICES=2
16          numactl -N 0 -l $@
17          ;;
18      3 ) # CPU 0, GPU 3
19          export CUDA_VISIBLE_DEVICES=3
20          numactl -N 0 -l $@
21          ;;
22      4 ) # CPU 1, GPU 4
23          export CUDA_VISIBLE_DEVICES=4
24          numactl -N 1 -l $@
25          ;;
26      5 ) # CPU 1, GPU 5
27          export CUDA_VISIBLE_DEVICES=5
28          numactl -N 1 -l $@
29          ;;
30      6 ) # CPU 1, GPU 6
31          export CUDA_VISIBLE_DEVICES=6
32          numactl -N 1 -l $@
33          ;;
34      7 ) # CPU 1, GPU 7
35          export CUDA_VISIBLE_DEVICES=7
36          numactl -N 1 -l $@
37          ;;
38  esac
```

※ CPU に分散配置するラッパースクリプト例

mpiwrap-alt.sh：ランク０から順に、図の順番で CPU、GPU を交互に割当てます。



```
#!/bin/bash

echo LANK=${OMPI_COMM_WORLD_LOCAL_RANK}

case ${OMPI_COMM_WORLD_LOCAL_RANK} in
    0 ) # CPU 0, GPU 0
        export CUDA_VISIBLE_DEVICES=0
        numactl -N 0 -l $@
        ;;
    1 ) # CPU 1, GPU 4
        export CUDA_VISIBLE_DEVICES=4
        numactl -N 1 -l $@
        ;;
    2 ) # CPU 0, GPU 1
        export CUDA_VISIBLE_DEVICES=1
        numactl -N 0 -l $@
        ;;
    3 ) # CPU 1, GPU 5
        export CUDA_VISIBLE_DEVICES=5
        numactl -N 1 -l $@
        ;;
    4 ) # CPU 0, GPU 2
        export CUDA_VISIBLE_DEVICES=2
        numactl -N 0 -l $@
        ;;
    5 ) # CPU 1, GPU 6
        export CUDA_VISIBLE_DEVICES=6
        numactl -N 1 -l $@
        ;;
    6 ) # CPU 0, GPU 3
        export CUDA_VISIBLE_DEVICES=3
        numactl -N 0 -l $@
        ;;
    7 ) # CPU 1, GPU 7
        export CUDA_VISIBLE_DEVICES=7
        numactl -N 1 -l $@
        ;;
esac
```

5.6. コンテナの実行方法

本項ではコンテナを利用してジョブを実行する場合の、ジョブスクリプトについて説明します。本システムにおけるコンテナの利用方法の概要は、「1.5.4 コンテナ利用」を参照ください。

5.6.1. コンテナ実行の概要

コンテナの実行をする上で最低限把握しておくべき内容について説明します。標準的な環境のコンテナとして、centos のイメージファイル(centos.sif)を例としています。

➢ 実行コマンド

コンテナの実行は、exec サブコマンドを指定して実施します。実行例として、centos.sif コンテナイメージ内の hostname コマンドを実行する場合には以下のコマンドを実行します。

```
$ singularity exec centos.sif hostname
```

コマンドの書式は以下の通りです。

```
$ singularity exec <イメージファイル名> <コンテナ内の実行コマンド>
```

指定するコマンドは、**コンテナ内の実行コマンドとなっている点にご注意ください。**コマンドがパス指定なしであれば、コンテナ内の PATH 環境変数で探索されたコマンドが実行されます。パス指定有りの場合も、絶対パスはコンテナ内のファイル構造に従います。

➢ 環境変数

コンテナ外で定義した環境変数は、基本的にはコンテナ内にも引き継がれます。ただし、build 時などにコンテナ側で明示的に定義されている環境変数は、コンテナ側の定義に従います。

コンテナ側で定義されている環境変数を上書きする場合には、--env オプションによる個別の指定や、--env-file オプションによる一括の指定でコンテナ内に渡すことが可能です。

・--env オプションによる個別指定

```
$ singularity exec --env MYVAR="My Value!" centos.sif myprog.exe
```

・--env-file オプションによる一括指定

```
$ cat myenvfile
MYVAR="My Value!"
$ singularity exec --env-file myenvfile centos.sif myprog.exe
```

環境変数に関する詳細な内容は、下記の公式ドキュメントを参照ください。

https://sylabs.io/guides/3.7/user-guide/environment_and_metadata.html

➢ ホスト OS のマウント

コンテナ内からホスト OS のファイルシステムの read/write を行いたい場合には、ホスト OS の特定のディレクトリを bind マウントすることで利用可能です。オプション指定なしでも、**下記のディレクトリは標準でマウントされており、コンテナ内でも同じパスで利用が可能です。**

ホームディレクトリ：/sqfs/home/(利用者番号)

テンポラリ領域　　：/tmp

例として、コンテナ内からホスト OS の home ディレクトリに置かれたプログラム(a.out)を実行する場合のコマンドは以下となります。

```
$ singularity exec centos.sif ./a.out
```

※ 上記例では、カレントディレクトリが、コンテナ内で home に移動しています。

ホスト OS の特定のディレクトリをマウントする場合には、--bind オプションを利用します。--bind オプションの書式は以下となります。

--bind <ホスト OS のパス>:<コンテナ内のパス>:<モード>

コンテナ内のパス並びにモード(ro/rw)は省略可能です。省略した場合は、コンテナ内のパスは、ホスト OS のパスと同じパスで read/write でマウントされます。

例として、拡張領域上のディレクトリに置かれたプログラム(a.out)を実行する場合のコマンドは以下となります。

```
$ cd /sqfs/work/(グループ名)/(利用者番号)
$ singularity exec --bind `pwd` centos.sif ./a.out
```

※ 上記例では、コンテナ外での環境変数 PWD が、コンテナ内へも引き継がれており、コンテナ内のカレントディレクトリが、拡張領域上のディレクトリになっています。

ホスト OS のマウントに関するする詳細な内容は、下記の公式ドキュメントを参照ください。

https://sylabs.io/guides/3.7/user-guide/bind_paths_and_mounts.html

5.6.2. 汎用 CPU 計算環境での実行方法

汎用 CPU 計算環境で、１ノード内でスレッド並列プログラム実行を想定したジョブスクリプトのサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 24 時間）
- 総 CPU コア数　　：76（=38 コア×2CPU×1 ノード）

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -v OMP_NUM_THREADS=76
7
8   #------- Program execution -----------
9
10  cd $PBS_O_WORKDIR
11  singularity exec --bind `pwd` image.sif ./a.out
12
```

- 6 行目

  並列実行数を指定します。

- 11 行目

  イメージファイルを指定して、singularity exec コマンドを実行します。

利用時の注意点

実行スクリプトにて、「OMP_NUM_THREADS」を忘れずに指定してください。

「OMP_NUM_THREADS」を指定しない場合、あるいは間違った値を指定してしまった場合、意図しない並列数での実行となる可能性があります。ご注意ください。

5.6.3. GPGPU 計算環境での実行方法

GPGPU 計算環境で、１ノード内でスレッド並列プログラム実行を想定したジョブスクリプトのサンプルです。

- キュー　　：SQUID
- グループ　　：G01234
- 経過時間　　：30 分（未指定の場合は 24 時間）
- スレッド数　　：76

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q SQUID
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -l gpunum_job=8
7   #PBS -v OMP_NUM_THREADS=76
8
```

```
9   #------- Program execution -----------
10
11  cd $PBS_O_WORKDIR
12  singularity exec --nv --bind `pwd` image.sif ./a.out
13
```

- 7 行目

  並列実行数を指定します。

- 12 行目

  イメージファイルを指定して、singularity exec コマンドを実行します。コンテナ内から GPGPU を利用可能とするため、--nv オプションを付加します。

利用時の注意点

実行スクリプトにて、「OMP_NUM_THREADS」を忘れずに指定してください。

「OMP_NUM_THREADS」を指定しない場合、あるいは間違った値を指定してしまった場合、意図しない並列数での実行となる可能性があります。ご注意ください。

6. アプリケーションの使い方

本システムでは、以下のアプリケーションを備えています。

6.1. アプリケーション一覧

- ISV アプリケーション

ISV アプリケーション	利用可能ホスト				
	squidhpc	squidhpda	cpu	vec	gpu
AVS/Express Developer	○	○	-	-	-
Gaussian	○	○	○	-	-
IDL	○	○	-	-	-

- OSS アプリケーション

OSS アプリケーション	利用可能ホスト				
	squidhpc	squidhpda	cpu	vec	gpu
ADIOS	○	○	○	-	-
miniforge	○	○	○		
GAMESS	○	○	○	-	-
Gnuplot	○	○	-	-	-
GROMACS	○	○	○	-	○
ImageMagick	○	○	-	-	-
LAMMPS	○	○	○	-	-
NcView	○	○	○	-	-
Octave	○	○	○	-	-
OpenFORM	○	○	○	-	-
ParaView	○	○	-	-	-
Quantum ESPRESSO	○	○	○	○	○
PHASE/0	○	○	○	○	-
Relion	○	○	○	-	-
VisIt	○	○	○	-	-

- 国のプロジェクト等で開発されたアプリケーション

国プロ アプリケーション	利用可能ホスト				
	squidhpc	squidhpda	cpu	vec	gpu
ABINIT-MP	○	○	○	○	-
HΦ	○	○	○	-	-
MODYLAS	○	○	○	-	-
NTChem	○	○	○	-	-
OpenMX	○	○	○	-	-
SALMON	○	○	○	-	-
SMASH	○	○	○	-	-

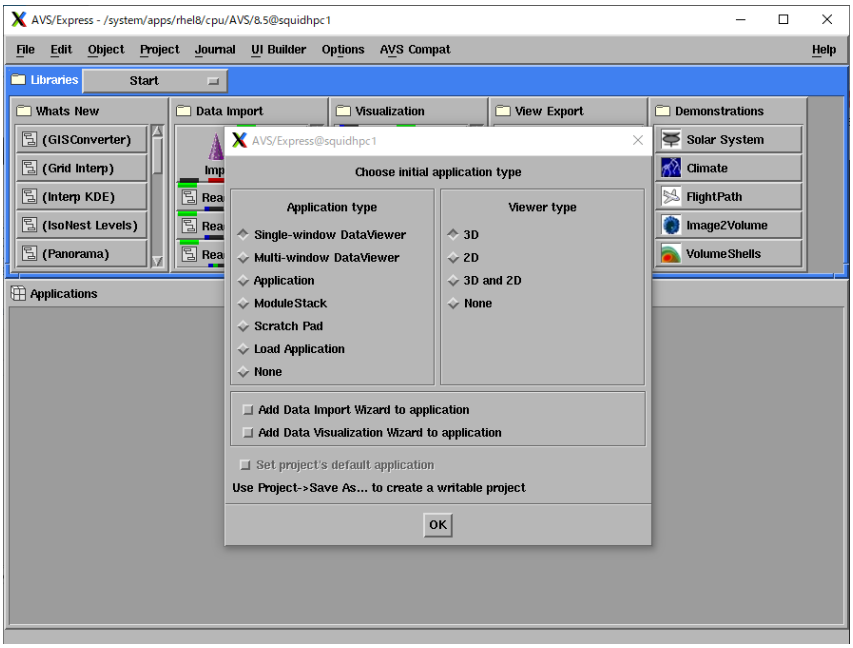
6.2. ISV アプリケーション利用方法

6.2.1. AVS/Express

可視化処理用アプリケーションである「AVS/Express」は、バージョン 8.5 を提供しております。SQUID における同時利用ユーザ数は 5 名となります。X ターミナルソフト等を利用し、フロントエンドにログイン後、以下コマンドを実行して AVS/Express を起動します。

```
$ module load BaseApp
$ module load AVSExpress/8.5
$ express
```

以下のように「AVS/Express」の操作画面が表示されます。



また、「AVS/Express」は SQUID でライセンスサーバを提供しております。AVS/Express をお手元の端末にインストールしてご利用いただく場合は、以下のライセンスサーバとポート番号を指定してください。

項	ライセンスサーバ	ポート番号
1	squidportal.hpc.cmc.osaka-u.ac.jp	XXXXX/tcp

※ライセンス番号は年度ごとに変わりますので管理者にお問い合わせください。

ライセンスはフローティングライセンスで、ライセンス数は 5 です。ライセンスが不足している場合は、起動時にターミナルに以下メッセージが表示されます。

```
Could not get license from server: license limit exceeded
```

6.2.2. Gaussian

量子化学計算プログラムである「Gaussian」は、バージョン g16 を提供しております。利用する場合は、Gaussian 用アプリケーショングループへの登録申請を、センターへご連絡いただく必要があります。

プログラムはバッチジョブで実行してください。以下はサンプルプログラムを実行する例です。

```
#!/bin/bash

#PBS -q SQUID
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00

module load BaseApp
module load Gaussian

newgrp gaussian

cd $PBS_O_WORKDIR
g16root/g16/g16 <  入力ファイル  >& result
```

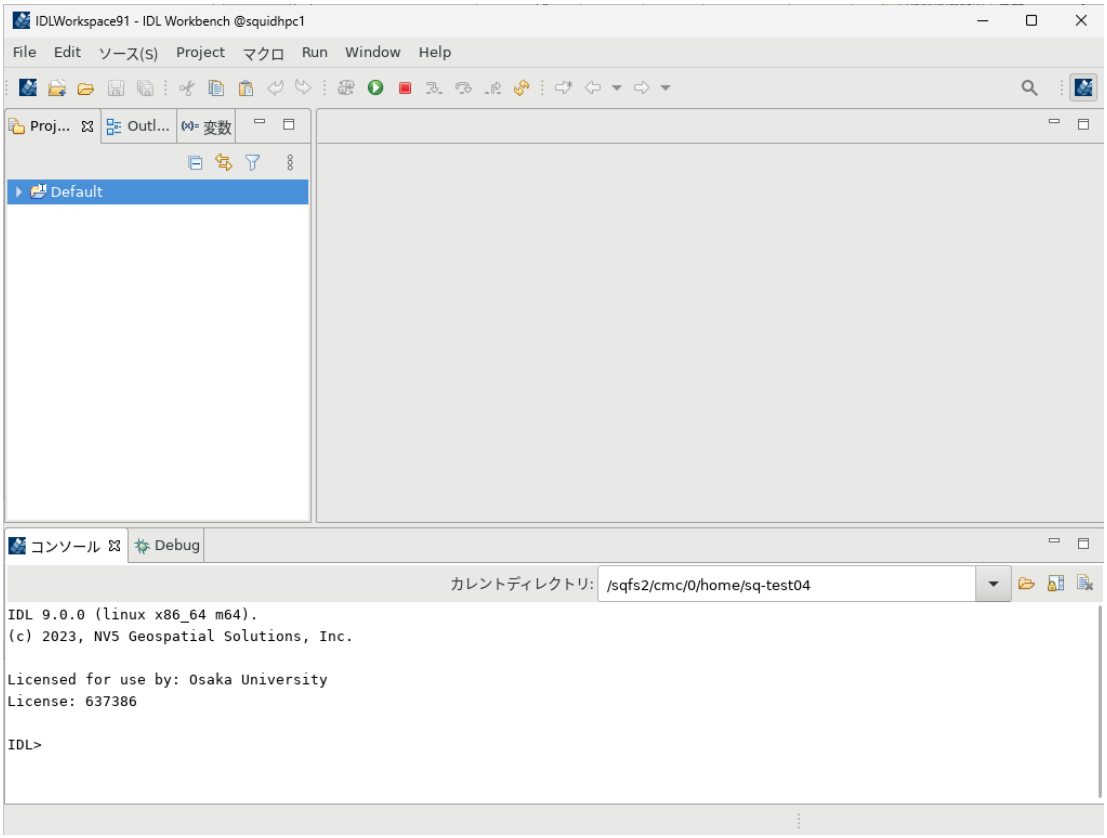
6.2.3. IDL

「IDL」はバージョン 8.8、9.0 を提供しております。同時利用ユーザ数は 4 名となります。X ターミナルソフト等を利用し、フロントエンドにログイン後、以下コマンドを実行して「IDL」を起動します。

```
$ module load BaseApp
```

```
$ module load IDL/9.0
$ idlde
```

以下のように IDL の操作画面が表示されます。



ライセンスはフローティングライセンスで、ライセンス数は 4 です。ライセンスが不足している場合は、以下メッセージが記載されたダイアログボックスが表示されます。

```
Unable to license IDL.
```

6.3. OSS アプリケーション利用方法

6.3.1. ADIOS

フロントエンドにログイン後、以下を実行します。

```
$ module load BaseApp
$ module load adios/1.13.1
```

※利用方法については準備中になります。お待ちください。

6.3.2. GAMESS

プログラムはバッチジョブで実行してください。以下はサンプルプログラムを実行する例です。

特に何も指定しない場合は、スクラッチディレクトリとして/sqfs/work/(グループ名)/(利用者番号)/scr が生成されます。

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 2
#PBS -T intmpi

module load BaseApp
module load GAMESS/v2020.2

cd $PBS_O_WORKDIR
rungms exam01.inp 00 24
```

スクラッチディレクトリを指定する場合は、プログラム実行前に下記の環境変数を定義します。

```
$ export SCR=<スクラッチディレクトリ>
$ export USERSCR=<スクラッチディレクトリ>
```

6.3.3. Gnuplot

フロントエンドにログイン後、以下を実行します。

```
$ gnuplot
```

6.3.4. GROMACS

GROMACS は、単精度/倍精度の違い、シングル版/MPI 版の違い、GPU 対応有無で、モジュール名とバイナリ名が異なります。それぞれのモジュール名とバイナリ名は以下の通りです。

種別	cpu (GPU 非対応版)		gpu (GPU 対応版)	
	モジュール名	バイナリ名	モジュール名	バイナリ名
単精度シングル	gromacs/2022.6	gmx	gromacs/2021.2.GPU	gmx
単精度 MPI	gromacs/2022.6	gmx_mpi	gromacs/2021.2mpi.GPU	gmx_mpi

倍精度シングル	gromacs/2022.6	gmx_d	–	–
倍精度 MPI	gromacs/2022.6	gmx_mpi_d	–	–

以下にサンプルプログラムの実行例をいくつか記載します。プログラムはバッチジョブで実行してください。

<CPU ノード:単精度シングル版>

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 1
#PBS -j o

module load BaseApp
module load gromacs/2022.6
export OMP_NUM_THREADS=64

cd $PBS_O_WORKDIR
gmx grompp -f MD1.mdp -c ./replace_po_mg.gro -p ./topol.top -o md1.tpr
gmx mdrun -deffnm md1 -ntomp 64
```

<CPU ノード:倍精度 MPI 版>

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 2
#PBS -T intmpi
#PBS -v OMP_NUM_THREADS=1

module load BaseApp
module load gromacs/2022.6
```

```
cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 152 -ppn 76 gmx_mpi_d mdrun -ntomp 1 -s ch36.tpr
```

<GPU ノード:単精度 MPI 版>

```
#!/bin/bash
#PBS -q SQUID
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 2
#PBS -T openmpi
#PBS -l gpunum_job=8
#PBS -v NQSV_MPI_MODULE=BaseGCC/2021:cuda/11.2
#PBS -v OMP_NUM_THREADS=6

module load BaseApp
module load gromacs/2021.2mpi.GPU

cd $PBS_O_WORKDIR
export GMX_MPI=`which gmx_mpi`

mpirun ${NQSV_MPIOPTS} -np 2 -npernode 1 ${GMX_MPI} mdrun -s poly-ch2.tpr
```

6.3.5. ImageMagick

ImageMagick は画像の操作や表示をするためのソフトウェアです。

以下に画像の拡張子を変更(JPG→PNG)する方法を例として記載します。

```
$ convert sample.jpg sample.png
```

6.3.6. LAMMPS

LAMMPS は、オープンソースの分子動力学アプリケーションです。

導入パッケージは以下のとおりです。

```
ASPHERE,BODY,CLASS2,COLLOID,COMPRESS,CORESHELL,DIPOLE,GRANULAR,KSPACE,
MANYBODY,MC,MISC,MOLECULE,MPIIO,OPT,PERI,POEMS,PYTHON,QEQ,REPLICA,RIGID,
SHOCK,SNAP,SPIN,SRD,VORONOI,USER-ATC,USER-AWPMD,SER-BOCS,USER-CGDNA,
USER-CGSDK,USER-COLVARS,USER-DIFFRACTION,USER-DPD,USER-DRUDE,USER-EFF,
USER-FEP,USER-INTEL,USER-LB,USER-MANIFOLD,USER-MEAMC,USER-MESODPD,
```

```
USER-MGPT,USER-MISC,USER-MOFFF,USER-NETCDF,USER-OMP,USER-PHONON,USER-QTB,
USER-REAXC,USER-SMTBQ,USER-SPH,USER-TALLY,USER-UEF
```

バッチジョブで実行してください。ジョブスクリプト例は下記の通りです。

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 1
#PBS -v OMP_NUM_THREADS=38

module load BaseApp
module load lammps/29-Oct-20
cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 2 lmp < ./in.lj
```

6.3.7. Miniforge

構築済みの仮想環境一覧を表示します。

```
$ conda info -e
# conda environments:
#
base /sqfs/work/(グループ名)/(利用者番号)/miniforge
```

新規に、test-env という仮想環境をユーザ領域に作成する際の手順を示します。
まず、仮想環境の生成先として、work 領域配下の conda_env ディレクトリを指定します。
（特に何も指定しなかった場合、home 領域配下の.conda ディレクトリで仮想環境が生成され、容量オーバーとなる可能性があります）

```
$ conda config --add envs_dirs /sqfs/work/(グループ名)/(利用者番号)/conda_env
$ conda config --add pkgs_dirs /sqfs/work/(グループ名)/(利用者番号)/conda_pkg
```

仮想環境 test-env を生成します。

```
$ conda create --name test-env python=3.8
```

仮想環境一覧を表示すると、test-env が生成されていることがわかります。

```
$ conda info -e
# conda environments:
#
test-env * /sqfs/work/(グループ名)/(利用者番号)/conda_env/test-env
base /sqfs/work/(グループ名)/(利用者番号)/miniforge
```

仮想環境 test-env を有効にします。

```
$ conda activate test-env
```

仮想環境 test-env に Python のライブラリ numpy をインストールします。
計算ノード上から外部への通信はできませんので、必ずログインノードで実施してください。

```
$ conda install numpy
```

仮想環境 test-env を無効にします。

```
$ conda deactivate
```

miniforge で Pytorch を利用する方法は以下の通りです。
仮想環境 pytorch-env を作成し有効化します。

```
$ conda create --name pytorch-env python=3.12
$ conda activate pytorch-env
```

Pytorch の公式ページを参考に、Conda Package かつ CUDA11 系に対応した Pytorch をインストールします。
2024 年 8 月時点では以下のようになります。

```
$ conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia
```

SQUID 計算ノードでの利用方法は以下の通りです。
仮想環境「test-env」を有効化し test.py を SQUID GPU ノード 1 ノードで実行する場合、以下のようなジョブスクリプトを作成してください。

```
#!/bin/bash
#PBS -q SQUID
#PBS --group=(グループ名)
#PBS -l elapstim_req=1:00:00,gpunum_job=8
cd $PBS_O_WORKDIR
```

```
source ~/.bashrc
conda activate test-env
python test.py
```

2 ノードで実行する場合は、以下のようなジョブスクリプトを作成してください

```
#!/bin/bash
#PBS -q SQUID
#PBS --group=(グループ名)
#PBS -b 2
#PBS -l gpunum_job=8
#PBS -l elapstim_req=01:00:00
cd $PBS_O_WORKDIR
source ~/.bashrc
conda activate test-env
echo "===================="
MASTER_NODE=`head -1 ${PBS_NODEFILE}`
USE_PORT=12321
HOSTNAME=`hostname -s`
echo "------------------"
echo "HOSTNAME =[${HOSTNAME}] MASTER_NODE=[${MASTER_NODE}]
USE_PORT=[${USE_PORT}]"
echo "===================="
python -m torch.distributed.launch --nnodes=2 --nproc-per-node=8 --rdzv-id=100 --
rdzv-backend=c10d --rdzv-endpoint=${MASTER_NODE}:${USE_PORT} test.py
```

6.3.8. Octave

フロントエンドにログイン後、以下を実行します。

```
$ module load BaseApp
$ module load octave/10.1.0
$ octave
```

6.3.9. OpenFOAM

OpenFOAM は流体/連続体シミュレーションプラットフォームです。

バッチジョブで実行してください。ジョブスクリプト例は下記の通りです。

<シリアル版>　※ソフトウェア付属のチュートリアルである pitzDaily を使用しています。

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 1

module load BaseApp
module load OpenFOAM/v2012

cd $PBS_O_WORKDIR
blockMesh
simpleFoam
```

<MPI 版>

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 2

module load BaseApp
module load OpenFOAM/v2012

cd $PBS_O_WORKDIR
blockMesh
cp 0.orig 0 -r
decomposePar
mpirun ${NQSV_MPIOPTS} -np 152 interFoam -parallel
```

6.3.10. Quantum ESPRESSO

Quantum ESPRESSO は、第一原理計算に基づく電子構造計算と材料モデリングを行うための OSS です。

ベクトルノード版で提供されているモジュールは、平面波基底 SCF 計算パッケージ(PWscf)のみです。ジョブスクリプト例は下記の通りです。

```
#!/bin/bash
#-------------- qsub option qsub option ----------------------
#PBS -q DBG
#PBS --group=sq-test
#PBS -l elapstim_req=00:10:00
#PBS --venode=2
#PBS -T necmpi
#-------------- Program execution Program execution ---------

module load BaseApp
module load QuantumESPRESSO/6.4.1.VEC

export LM=`which pw.x`
cd $PBS_O_WORKDIR
mpirun –venode -np 16 -version /opt/nec/ve/bin/mpisep.sh ${LM} ¥
                        "-npool 2 -nband 1 -ntg 1 -ndiag 4 -input ausurf.in"
```

6.3.11. PHASE/0

PHASE/0 は密度汎関数理論に基づく第一原理電子状態計算を行うための OSS です。2025 年 4 月現在、CPU ノード版、ベクトルノード版が提供されています。

ベクトルノード版で提供されているモジュールは、第一原理電子状態計算パッケージ(phase)のみとなります。ジョブスクリプト例は下記の通りです。

```
#!/bin/sh

#-------------- qsub option qsub option ----------------------
#PBS -q DBG
#PBS --group=sq-test
#PBS -l elapstim_req=00:10:00
#PBS --venode=1
#PBS -T necmpi
#-------------- Program execution Program execution ---------

module load BaseApp
```

```
module load PHASE0/2023.01.3D

export NMPI_SEPSELECT="3"
export NMPI_PROGINF="DETAIL"
export VE_PROGINF="DETAIL"
export VE_ACC_IO="1"

export LM=`which phase`
cd $PBS_O_WORKDIR

echo "Start at `date`"
mpiexec –venode -np 8 ${LM}
echo "End at `date`"
```

6.3.12. Paraview

フロントエンドで「ParaView」を使用する場合は、X ターミナルソフト等を利用し、フロントエンドへログイン後、以下を実行します。

```
$ module load BaseApp
$ module load ParaView/5.8.1
$ paraview
```

6.3.13. Relion

Relion はオープンソースの電子顕微鏡用画像処理ソフトです。フロントエンドへログイン後、以下を実行します。

```
$ module load BaseApp
$ module load relion/3.1.1
$ relion
```

6.3.14. Visit

VisIt はオープンソースの可視化ソフトです。フロントエンドへログイン後、以下を実行します。

```
$ module load BaseApp
$ module load VisItv/3.1.3
$ visit
```

6.4. 国のプロジェクト等で開発されたアプリケーションの利用方法

6.4.1. ABINIT-MP

ABINIT-MP は、フラグメント分子軌道（FMO）計算を高速に行えるソフトウェアです。専用 GUI の BioStation Viewer との連携により、入力データの作成～計算結果の解析が容易に行えます。4 体フラグメント展開（FMO4）による 2 次摂動計算も可能です。

CPU ノード版、ベクトルノード版が用意されていますが、使えるモジュールに制限があります。

ノード種別	F 関数あり		F 関数なし	
	SMP 対応	SMP 非対応	SMP 対応	SMP 非対応
cpu	abinitmp_smp-fint	abinitmp-fint	abinitmp_smp	abinitmp
vec	-	-	-	abinitmp-vec

バッチジョブで実行してください。ジョブスクリプト例は下記の通りです。

<CPU ノード>

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 1
#PBS -T intmpi

#PBS -v NUM_PROCS=19
#PBS -v NUM_THREADS=4
#PBS -v OMP_NUM_THREADS=${NUM_THREADS}
#PBS -v OMP_STACKSIZE=5G

#PBS -v BINARY_NAME=abinitmp_smp
#PBS -v FILE_NAME=gly5
#PBS -v NUM_CORE=_16n-144p-4t-smp-2020.1-intel
#PBS -v OUT_NAME=${FILE_NAME}${NUM_CORE}
ulimit –s unlimited
```

```
module load BaseApp
module load abinit-mp/2.4

cd ${PBS_O_WORKDIR}
mpirun ${NQSV_MPIOPTS} -np ${NUM_PROCS} ${BINARY_NAME} ¥
                                        < ${FILE_NAME}.ajf > ${OUT_NAME}
```

<ベクトルノード>

```
#!/bin/bash
#PBS -q SQUID
#PBS --group= <グループ名>
#PBS -T necmpi
#PBS -l elapstim_req=01:00:00
#PBS --venode=4
#PBS -v NUM_PROCS=40
#PBS -v FILE_NAME=gly5
#PBS -v AJF_NAME=${FILE_NAME}.ajf
#PBS -v VE_FORT5=${AJF_NAME}

module load BaseApp
module load abinit-mp/2.4.VEC

cd ${PBS_O_WORKDIR}
BINARY_NAME=`which abinitmp-vec`

mpirun -np ${NUM_PROCS} ${BINARY_NAME}
```

6.4.2. HΦ

オープンソースの数値厳密対角化法による有効模型ソルバーパッケージです。広汎な多体量子系の有効模型（多軌道ハバード模型、ハイゼンベルグ模型、近藤格子模型など）の基底状態及び低励起状態や励起スペクトル、有限温度における熱力学量を並列計算により求めることができます。

バッチジョブで実行してください。ジョブスクリプト例は下記の通りです。

```
#!/bin/bash
#PBS -q SQUID
```

```
#PBS -l cpunum_job=76
#PBS --group=<グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 1
#PBS -v OMP_NUM_THREADS=8

module load BaseApp
module load HPhi/3.5.1

cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 4 HPhi -s stan.in
```

6.4.3. MODYLAS

オープンソース（ライセンス許可制）の汎用古典分子動力学アプリケーションソフトウェアです。長距離静電相互作用の高速多重極展開法(FMM)による取り扱いを含め、ナノ分野・バイオ分野における分子動力学計算に必要な各種手法に対応しています。高効率な並列計算が可能です。

バッチジョブで実行してください。ジョブスクリプト例は下記の通りです。

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS --group=<グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 8
#PBS -T intmpi
#PBS -v OMP_NUM_THREADS=24
#PBS -v PARALLEL=24

module load BaseApp
module load MODYLAS/1.1.0

cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 8 modylas pyp111
```

6.4.4. NTChem

Gauss 型基底に基づいた量子化学計算アプリケーションソフトウェアです。多くのユーザに使っていただけるよう、様々な量子化学計算手法や機能が利用できます。大規模な分子に対して高効率な並列計算が可能です。

バッチジョブで実行してください。ジョブスクリプト例は下記の通りです。

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS –group=<グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -T intmpi
#PBS -b 1
#PBS -v OMP_NUM_THREADS=1

module load BaseApp
module load NTChem/12.0

cd ${PBS_O_WORKDIR}
./CH3CHO.sh
```

<CH3CHO.sh>

```
#!/bin/bash

export mol=CH3CHO
export curdir=`pwd`
export wrkdir=$curdir/ntchem

mkdir -p $curdir/ntchem

export nprocs=16
export mpirun="mpirun ${NQSV_MPIOPTS} -np $nprocs "
export sub=_mpi

export ntchem=$curdir/ntchem.sh
```

```
rm -f $wrkdir/$mol.* $wrkdir/INPUT
cd $wrkdir
cp -pr $curdir/${mol}.inp $wrkdir/INPUT
mpirun basinp${sub}.exe > $curdir/${mol}.log 2>&1
dlfind.exe < $curdir/${mol}.inp > $curdir/${mol}.out 2>&1
#
mv Coords.xyz $curdir/${mol}.xyz
cd $curdir
```

<ntchem.sh>

```
$mpirun mdint1${sub}.exe >> $curdir/$mol.log 2>&1
$mpirun scf${sub}.exe >> $curdir/$mol.log 2>&1
$mpirun scfgrad${sub}.exe >> $curdir/$mol.log 2>&1
```

6.4.5. OpenMX

オープンソースの第一原理計算アプリケーションソフトウェアです。原子局在基底と擬ポテンシャルを用いて、結晶・界面・溶液などの広範な物理系に対して電子状態計算を行います。

バッチジョブで実行してください。ジョブスクリプト例は下記の通りです。

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS --group=<グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -T intmpi
#PBS -b 2
#PBS -v OMP_NUM_THREADS=76

module load BaseApp
module load OpenMX/3.9

cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 2 openmx Methane.dat -nt 76
```

6.4.6. SALMON

オープンソースの光と物質の相互作用をターゲットにした第一原理計算アプリケーションソフトウェアです。時間依存密度汎関数理論に基づく実時間・実空間グリッド法を用いた光励起電子ダ

```
mpirun ${NQSV_MPIOPTS} -np 4 salmon < C2H2_gs.inp
```

6.4.7. SMASH

オープンソースの量子化学計算アプリケーションソフトウェアです。高速かつ高効率な並列計算手法により、ナノサイズ分子を分割せずに Hartree-Fock 法、DFT 法、MP2 法でエネルギーや構造最適化計算を実行することが可能です。

バッチジョブで実行してください。ジョブスクリプト例は下記の通りです。

```
#!/bin/bash
#PBS -q SQUID
#PBS -l cpunum_job=76
#PBS --group=<グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 1
#PBS -v OMP_NUM_THREADS=38

module load BaseApp
module load smash/2.2.0

cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 2 smash < b3lyp-energy.inp
```

7. ファイル転送方法　高度な使い方

7.1. Web ブラウザでのファイル転送

SQUID に WEB ブラウザでファイル転送する方法として、Nextcloud の環境を用意しています。

Nextcloud を経由して保存したファイルは、SQUID 内の以下のパスに保存され、フロントエンド環境や計算環境から利用することが可能です。

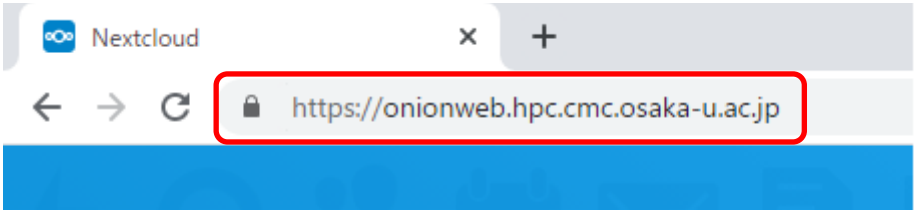
/sqfs/home/(利用者番号)/OnionWeb/

例: 利用者番号"user001"の場合　/sqfs/home/user001

7.1.1. ログイン

Web ブラウザを開き、以下の URL を入力します。

URL : https://onionweb.hpc.cmc.osaka-u.ac.jp



以下のログイン画面が表示されますので、利用者管理システムと同じユーザ名とパスワードを入力し、「ログイン」をクリックします。

ログインすると以下の画面になります。

ログイン後、自身のホーム領域が自動でマウントされ「SQUID ユーザホーム領域」として表示されます。

7.1.2. 基本的な使い方

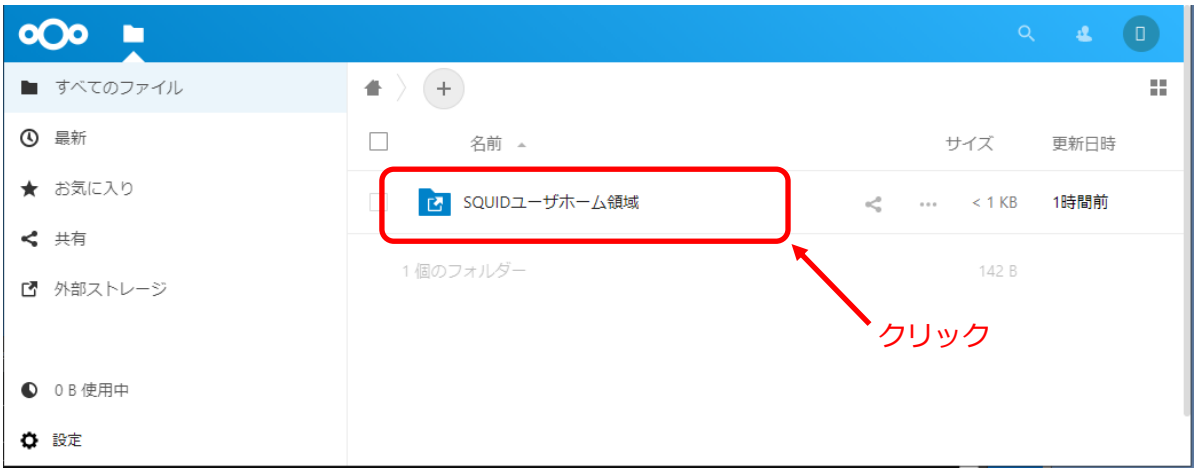
➢ **アップロード方法**

ファイルのアップロード先は、「SQUID ユーザホーム領域」、または後述する「7.1.7 外部ストレージの追加」にて追加したストレージのいずれかになります。

(1) ドラッグ＆ドロップによるアップロード

アップロードしたい場所を開き、アップロードしたいファイルをドラッグ＆ドロップします。
以下の例では「SQUID ユーザホーム領域」の直下にファイルをアップロードします。

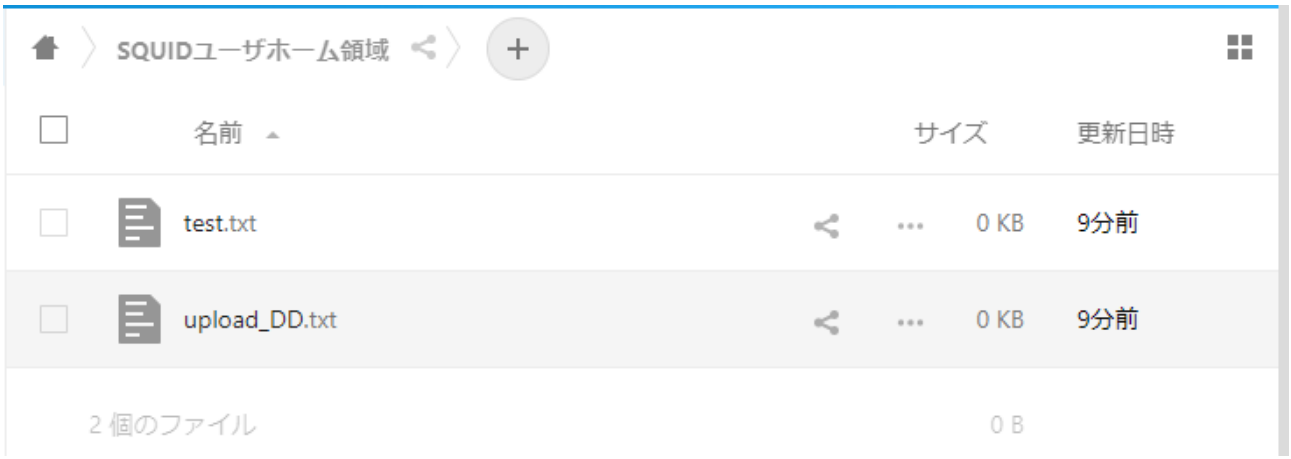
「SQUID ユーザホーム領域」をクリックします。



アップロードしたいファイルをドラッグ＆ドロップします。

アップロードが開始されると次のような進行状態バーが表示されますので少し待ちます。

ドラッグ＆ドロップしたファイルが表示されるとアップロード完了です。

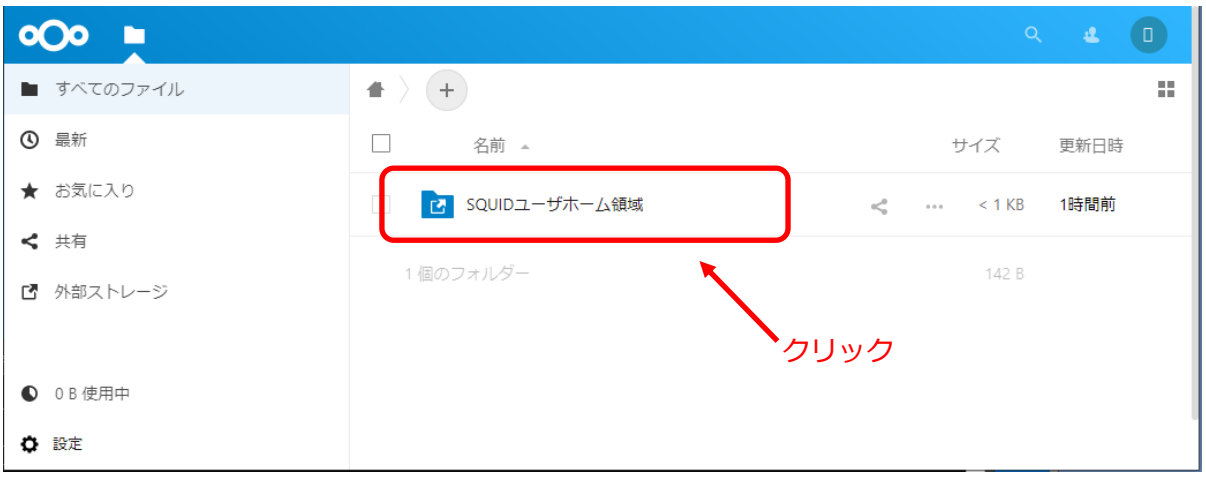


(2) 参照によるアップロード

アップロードしたい場所を開き、アップロードしたいファイルを選択します。

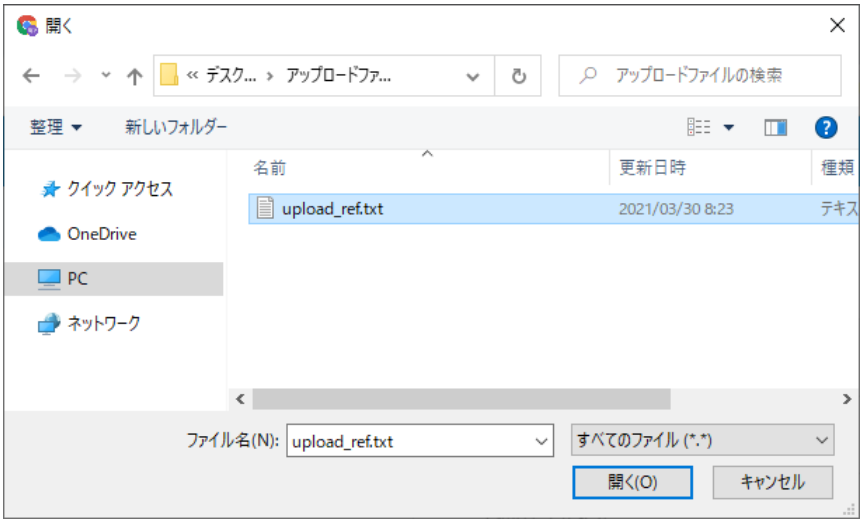
以下の例では「SQUID ユーザホーム領域」の直下にファイルをアップロードします。

「SQUID ユーザホーム領域」をクリックします。



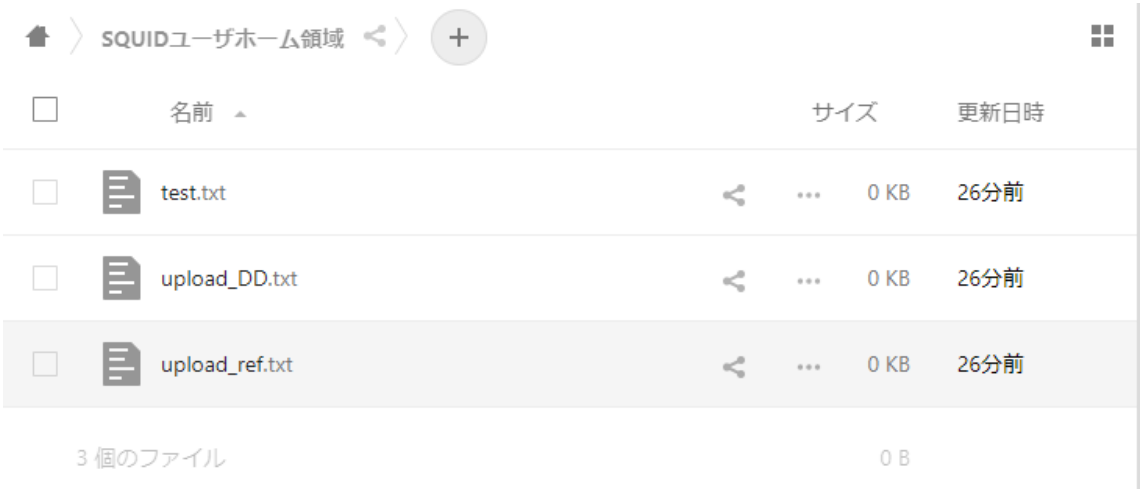
「⊕」をクリックし、表示される「ファイルをアップロード」をクリックします。

ファイル参照ダイアログが表示されますので、アップロードしたいファイルを選択し、
「開く」をクリックします



アップロードが開始されると次のような進行状態バーが表示されますので少し待ちます。

選択したファイルが表示されるとアップロード完了です。



➢ **ダウンロード方法**

(1) 単一ダウンロード

一つだけファイルをダウンロードする場合は以下の手順になります。

以下の例では「SQUID ユーザホーム領域」内の「test.txt」をダウンロードします。

ダウンロードしたいファイルの右側にある「…」をクリックし、表示される「ダウンロード」をクリックするとダウンロードが始まります。

ダウンロードファイルの保存先指定は使用している Web ブラウザの設定に依存します。

---

(2) 一括ダウンロード

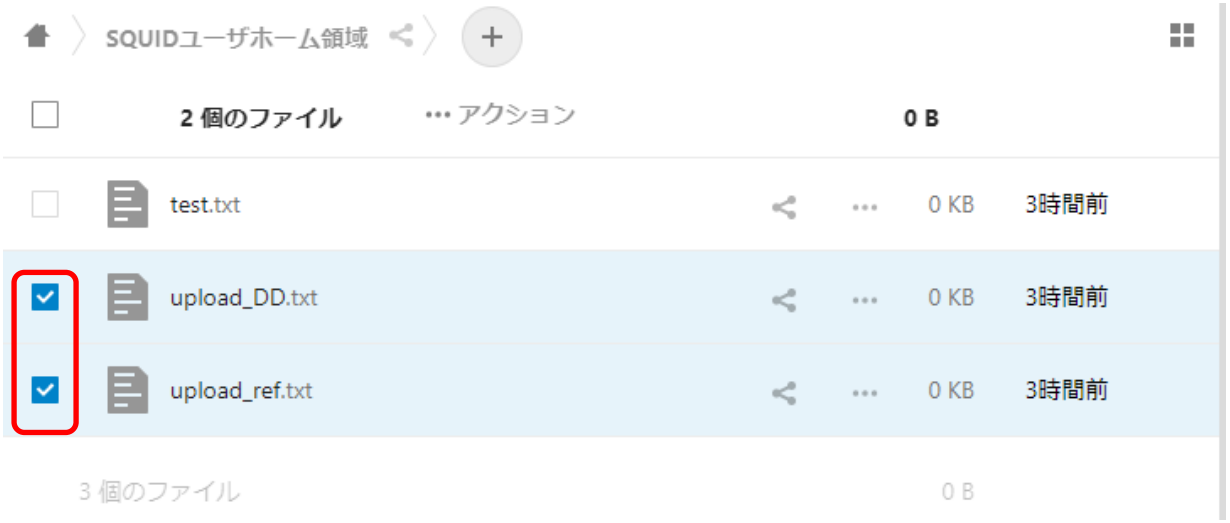
複数のファイルをまとめてダウンロードすることもできます。

まとめてダウンロードした場合は zip 形式で圧縮されたファイルがダウンロードされますので

別途解凍してください。

以下の例では「upload_DD.txt」と「upload_ref.txt」をまとめてダウンロードします。

ダウンロードしたいファイルの左側にあるチェックボックスにチェックを付けます。



ファイル名リストの上側にある「…アクション」をクリックし、表示される「ダウンロード」をクリックします。

ダウンロードファイルの保存先指定は使用している Web ブラウザの設定に依存します。

- **フォルダ名・ファイル名変更**

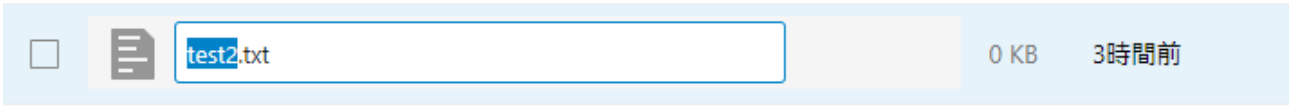
フォルダ名・ファイル名の変更は以下の手順になります。

以下の例では「test.txt」の名前を「test2.txt」に変更します。

名前変更したいフォルダ・ファイルの右側にある「…」をクリックし、表示される「名前の変更」

をクリックします。

対象のファイル名を編集できるようになりますので任意の名前に変更し、エンターキーを入力します。

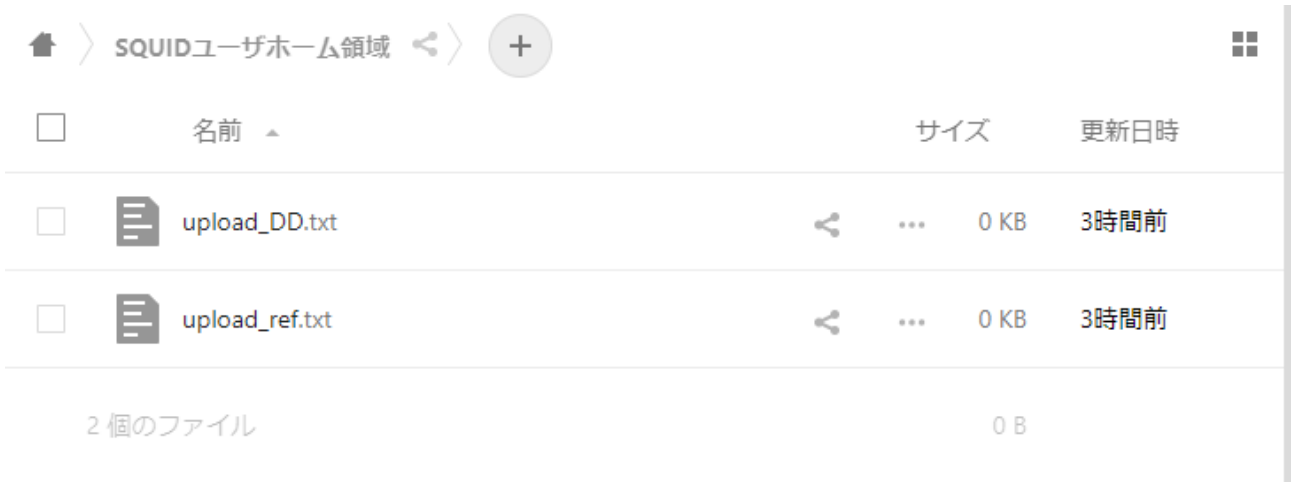


- **ファイルの削除**

フォルダ・ファイルの削除は以下の手順になります。以下の例では「test2.txt」を削除します。削除したいフォルダ・ファイルの右側にある「…」をクリックし、表示される「ファイルを削除」をクリックします。

削除が開始されると次のような進行状態バーが表示されますので少し待ちます。

ファイルの一覧画面から選択したフォルダ・ファイルが消えていれば削除完了となります。



- **お気に入り**

よく使うフォルダやファイルをお気に入りに登録しておくことで素早く参照することができます。

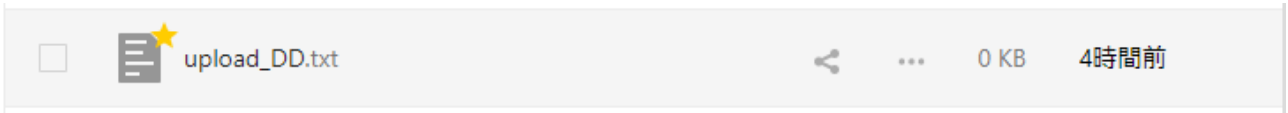
(1) お気に入り登録

フォルダ・ファイルのお気に入り登録は以下の手順になります。

以下の例では「upload_DD.txt」をお気に入りに登録します。

お気に入りに登録したいフォルダ・ファイルの右側にある「…」をクリックし、表示される「お気に入りに追加」をクリックします。

お気に入りに登録されるとアイコンの右上に「★」マークが付きます。



現在、お気に入りに登録しているフォルダ・ファイルを参照するには左メニューの「お気に入り」をクリックし、参照したいファイルをクリックします。

(2) お気に入りから削除

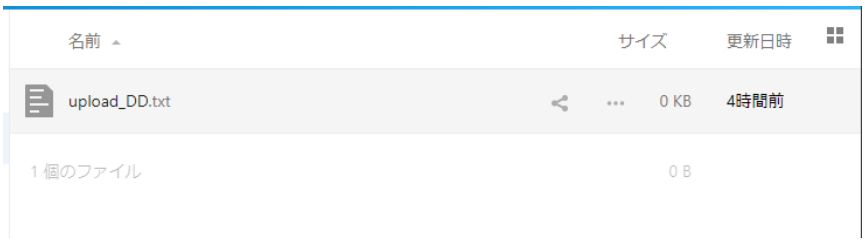
フォルダ・ファイルのお気に入りからの削除は以下の手順になります。

以下の例では「upload_DD.txt」をお気に入りから削除します。

左メニューの「お気に入り」をクリックします。

お気に入りから削除したいしたいファイル右側の「…」をクリックし表示される「お気に入りから削除」をクリックします。

先ほどのファイルから「★」マークが消えていればお気に入りから削除完了です。



7.1.3. フォルダ・ファイルの共有

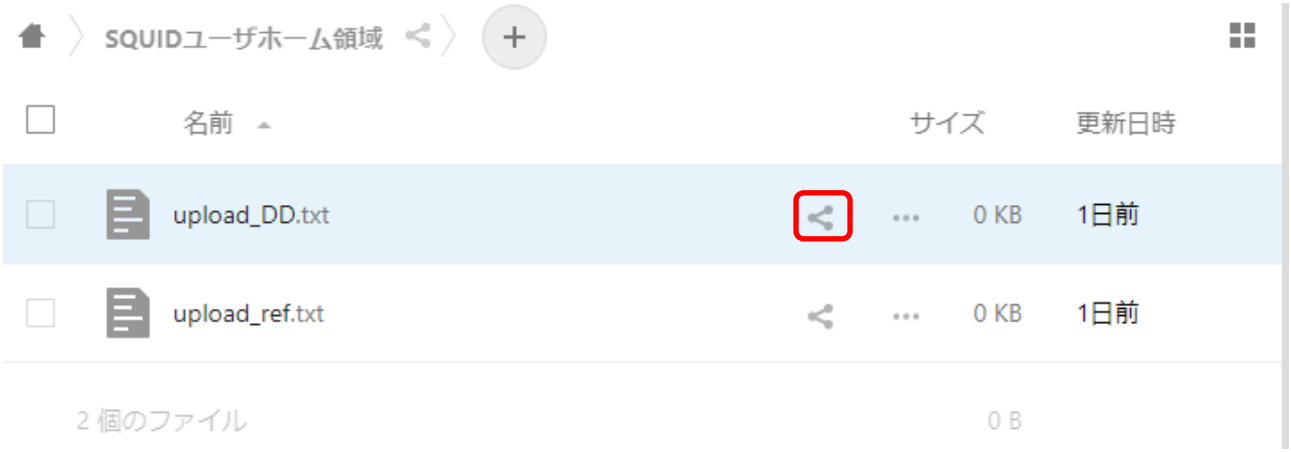
フォルダやファイルを指定して共有用の URL を発行することで、データ集約環境にアカウントを持たない人ともフォルダ・ファイルを共有することができます。

(1) フォルダ・ファイル共有設定（URL 共有）

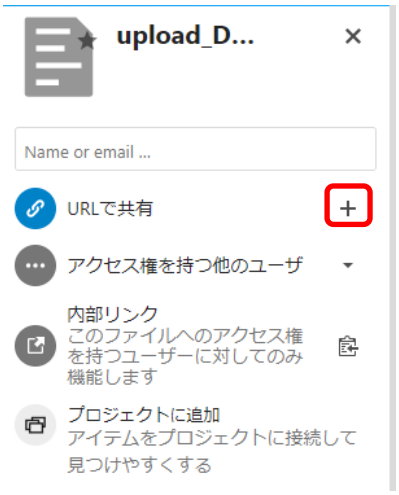
フォルダ・ファイルの共有は以下の手順になります。

以下の例では「upload_DD.txt」を共有します。

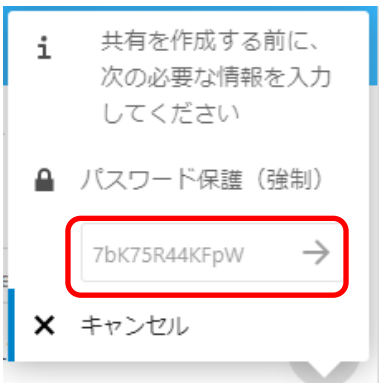
共有したいフォルダ・ファイルの右側にある共有マーク、「＜」をクリックします。



右側に以下のメニューが表示されますので「URL で共有」の右側「＋」をクリックします。

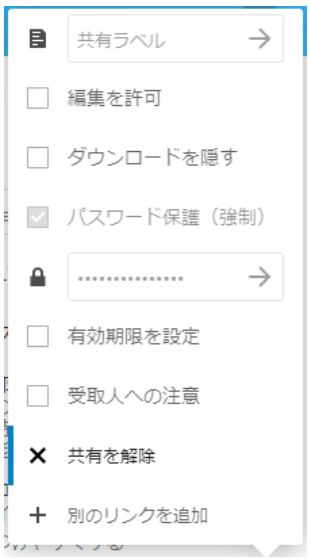


以下の画面が表示されますので「パスワード保護（強制）」に表示されるパスワードを控えるか、任意のパスワードを設定します。任意のパスワードを設定する場合は 12 文字以上にする必要があります。その後、パスワード右横の「→」をクリックします。



「URL で共有」の右側のアイコンが以下の様に変わりますので「…」をクリックします

以下のメニューが表示されますので、必要に応じて設定を変更します。
変更後メニュー外をクリックします。



共有ラベル：共有オーナが管理する上でのラベル
編集を許可：機能制限のため有効になりません。
ダウンロードを隠す：共有 URL 接続してもファイルが非表示になるため有効になりません。
パスワード保護（強制）：解除できません。任意に変更可能です。変更後は「→」をクリックしてください。
有効期限を設定：共有 URL の有効期限を設定できます。
受取人への注意：共有 URL の右上にメッセージを追加できます。

「URL で共有」の右側のクリップボードアイコンをクリックし、URL をコピーします。

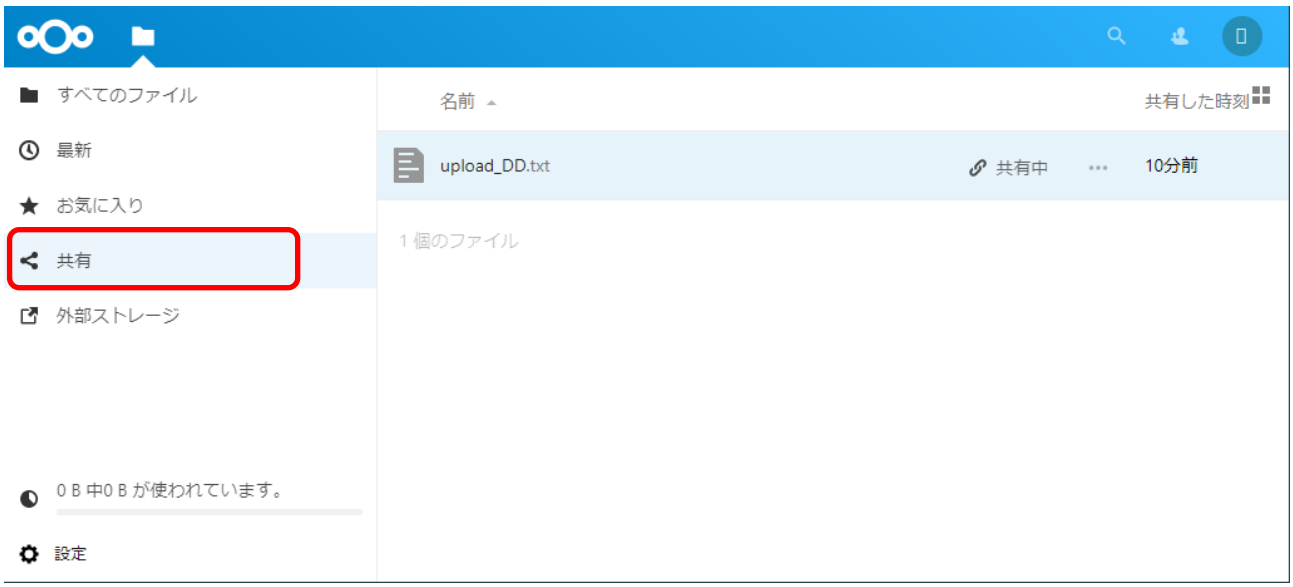
先に控えておいたパスワードとコピーした URL を、フォルダ・ファイルを共有したい人にメールなどで送ります。

(2) フォルダ・ファイル共有解除

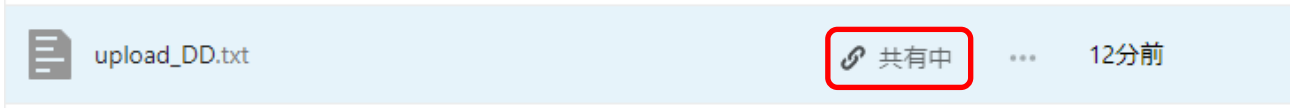
フォルダ・ファイルの共有解除は以下の手順になります。

以下の例では「upload_DD.txt」を共有解除します。

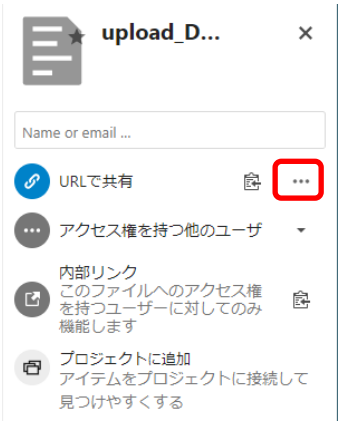
左メニューの「共有」をクリックします。



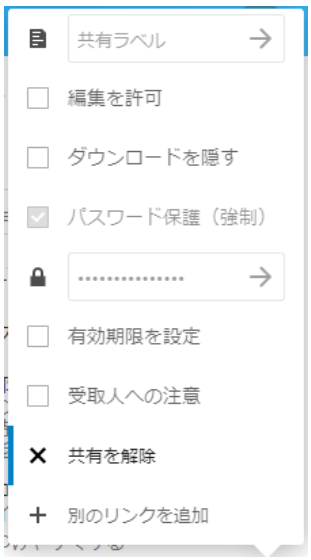
共有を解除したいフォルダ・ファイルの右側にある「共有中」をクリックします。



右側に以下のメニューが表示されますので「URL で共有」の右側「…」をクリックします。



以下のメニューが表示されますので、「共有を解除」をクリックします。

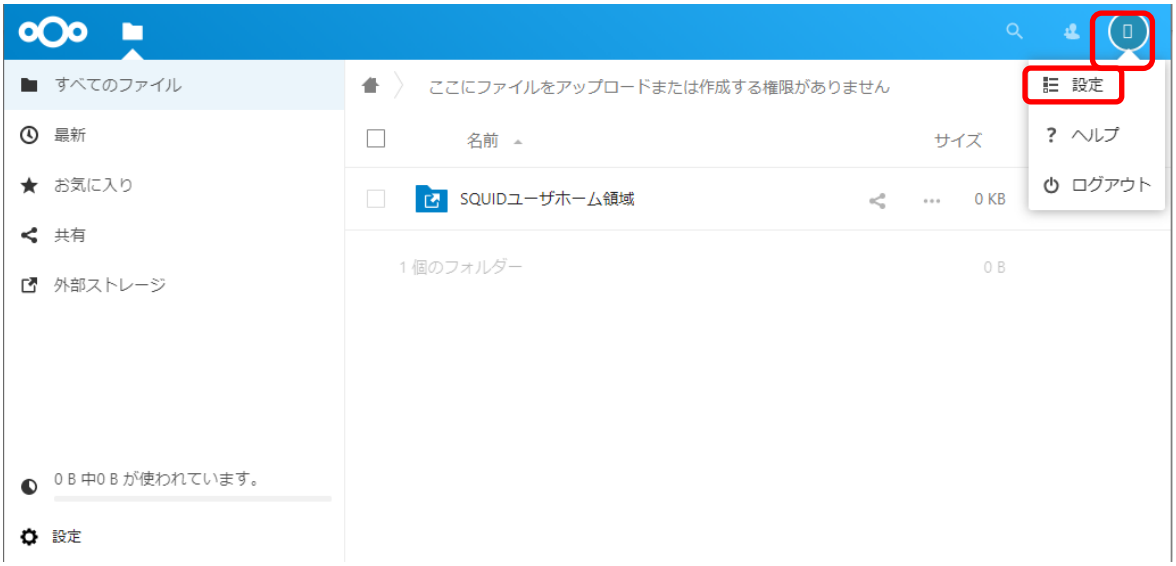


「URL で共有」の右側のアイコンが以下の様に変れば共有解除完了です。

7.1.4. 外部ストレージの追加

外部ストレージを追加することで Amazon S3 API に対応したストレージを、Nextcloud の環境から利用することが可能です。「1.3.1. 各環境の特徴」に記載されているアーカイブストレージも本手順にて Nextcloud から利用可能です。

画面右上のユーザアイコンをクリックし、表示されますメニューから「設定」をクリックします。

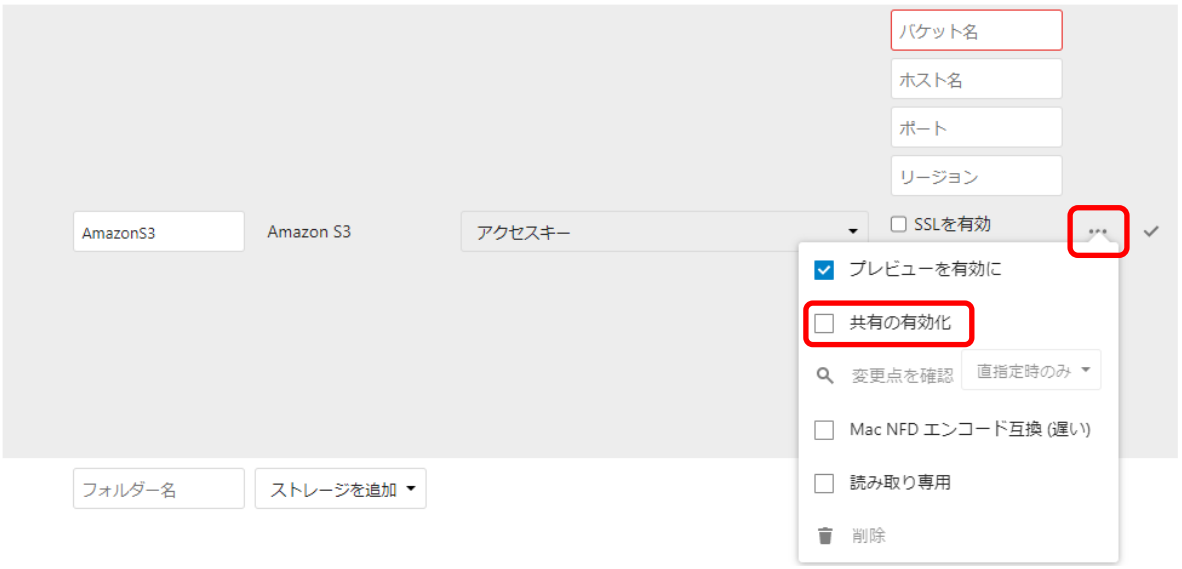


左メニューの「外部ストレージ」をクリックします。

「外部ストレージ」の「フォルダ名」に任意の名前を入力し、「ストレージを追加」➔「Amazon S3」をクリックします。

以下の設定項目が表示されますので、必要情報を入力します。

右側にある「…」をクリックし、「共有の有効化」にチェックを入れます。（共有しない場合は不要です）



設定完了後に右端の「✓」をクリックします。

7.1.5. アプリによる利用方法

Nextcloud 環境は、デスクトップクライアントアプリが用意されており、クライアントアプリ経由で利用することも可能です。クライアントアプリのインストール作業には再起動を伴います。インストールを行う際は編集中のファイルなど保存して実行してください。

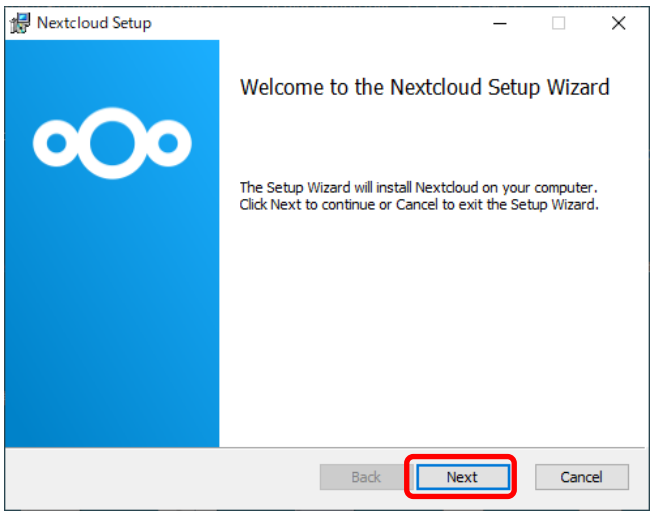
(1)インストール

以下の URL からインストーラをダウンロードします。

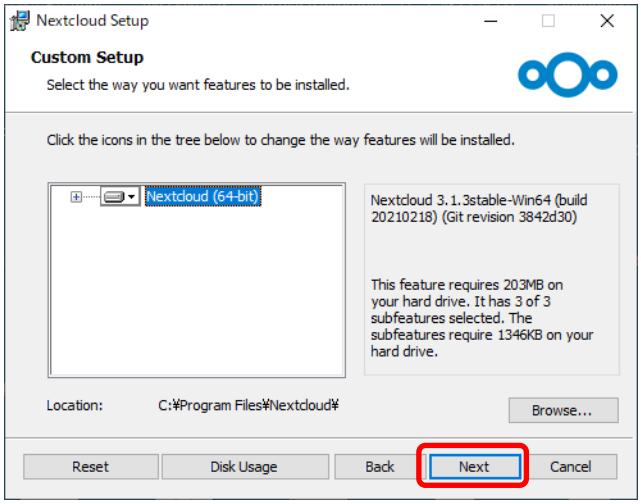
URL : https://nextcloud.com/install/#install-clients

ダウンロードしたインストーラを実行します。

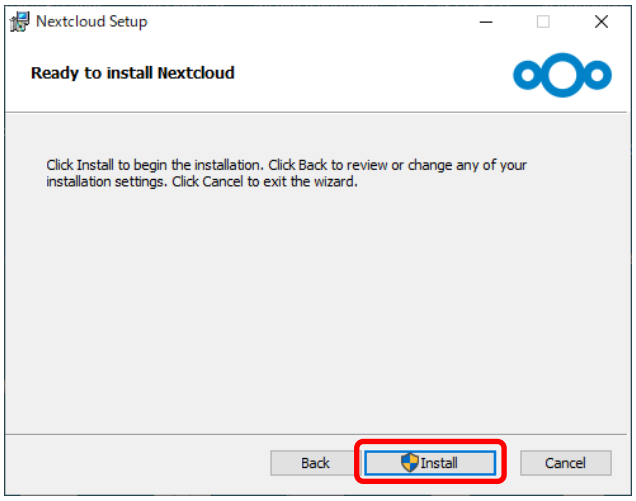
インストールが開始されますので「Next」をクリックします。



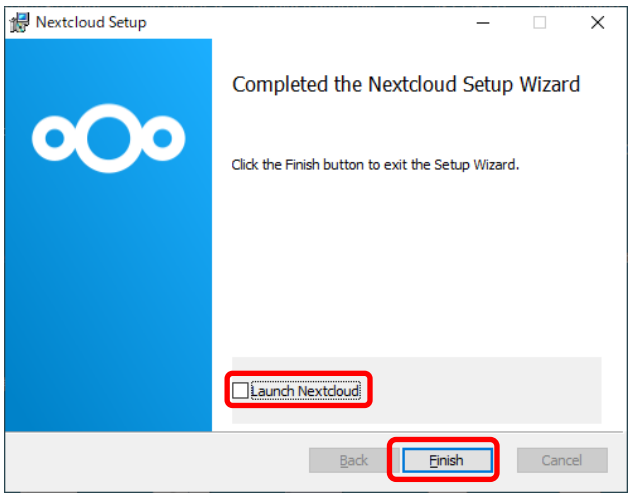
設定変更は特に必要ないため「Next」をクリックします。



インストールを実行するため「Install」をクリックします。

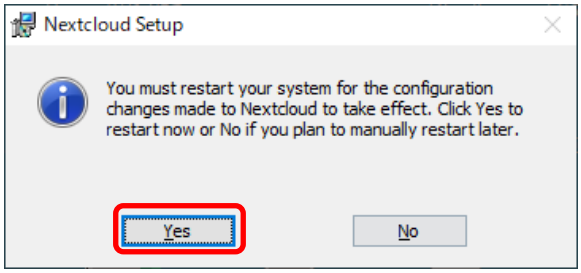


インストール完了後、「Launch Nextcloud」のチェックを外し、「Finish」をクリックします。



再起動を促すダイアログが表示されますので「Yes」をクリックし、パソコンを再起動します。

※編集中のファイルなどは事前に保存してください。



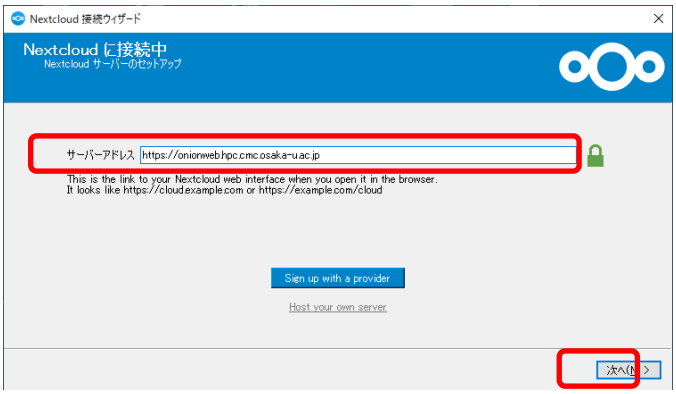
(2)設定

Nextcloud サーバと接続するための設定を行います。

パソコン再起動後、自動的に Nextcloud が起動し、次の画像が表示されますので「Log in to your Nextcloud」をクリックします。

「サーバーアドレス」に次の URL を入力し、「次へ」をクリックします。

URL : https://onionweb.hpc.cmc.osaka-u.ac.jp



以下の画面になり、Web ブラウザで認証許可する画面が自動的に起動します。

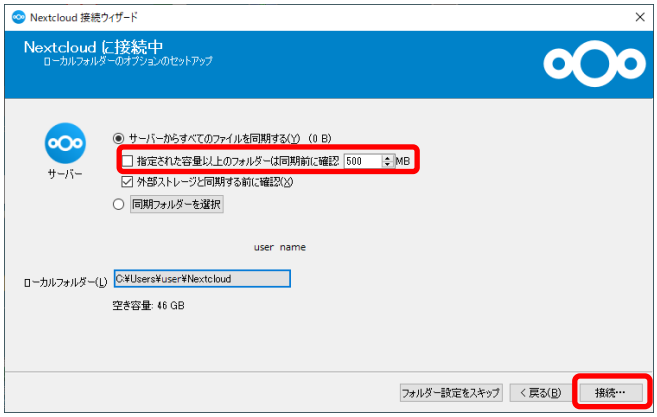
Web ブラウザにて以下の画面が表示されますので「ログイン」をクリックします。

以下のログイン画面が表示されますので、ユーザ名とパスワードを入力し、「ログイン」をクリックします。

以下の画面が表示されますので、「アクセスを許可」をクリックします。

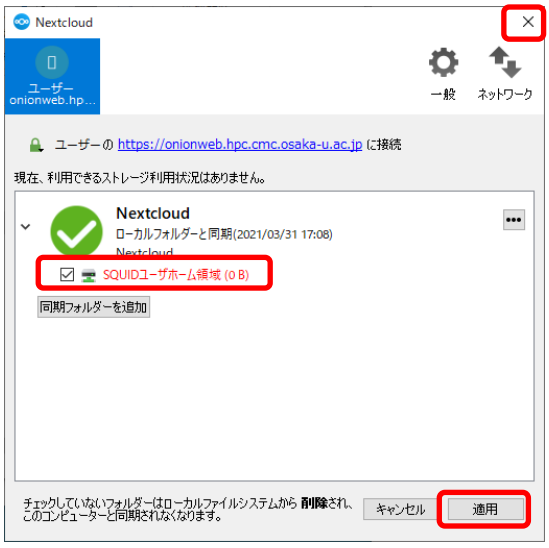
以下の画面が表示されますので、画面の指示に従い Web ブラウザを閉じます。

アプリケーションに戻り、以下の画面が表示されますので、「指定された容量以上のフォルダは同期前に確認」のチェックを外し、「接続…」をクリックします。

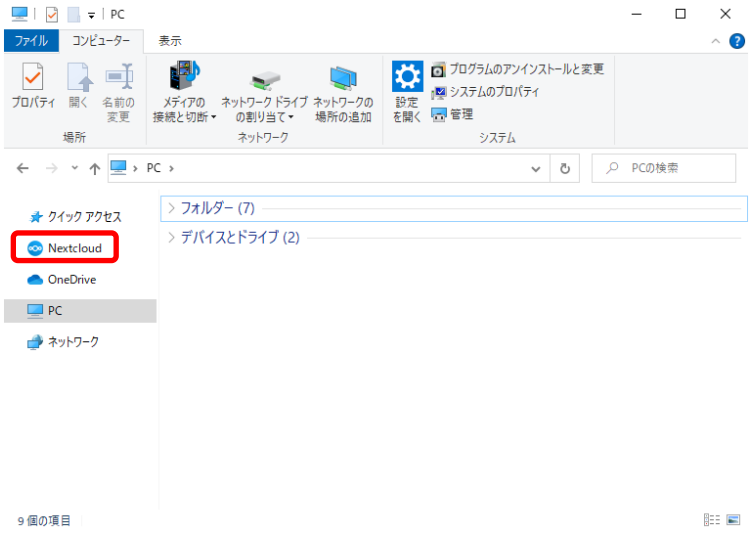


画面右下に Nextcloud の設定画面が表示されますので、ユーザ名の部分をクリックし、表示されるメニューの「設定」をクリックします。

以下の画面が表示されますので、「SQUID ユーザホーム領域」にチェックを付け、「適用」をクリックします。しばらく待つと同期が完了しますので「×」で閉じます。

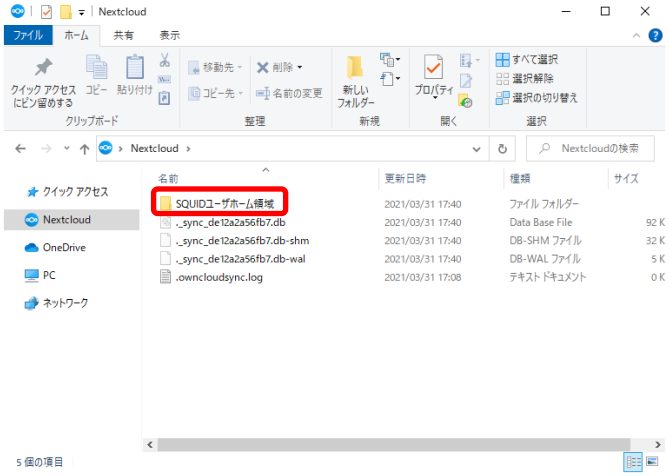


エクスプローラを開くと左側に「Nextcloud」が追加されていますのでクリックします。



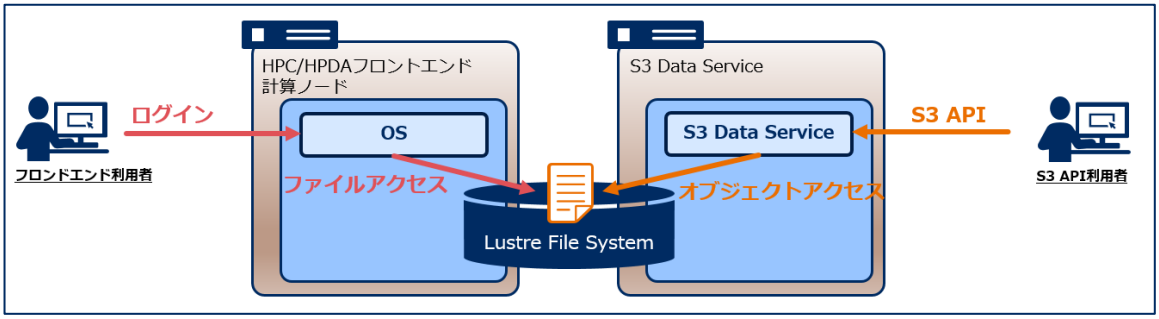
「.」から始まるファイル名のファイルは移動や削除しないでください。

「SQUID ユーザホーム領域」を開きます。ファイルの読み書きを行えます。



7.2. S3 API でのファイル転送

SQUID に S3 API を利用してファイル転送する方法として、S3 Data Service の環境を用意しています。S3 Data Service は、SQUID 上のファイルに対して、UNIX 的なファイルアクセスと S3 API からのオブジェクトアクセスを相互に行う環境を提供します。



これにより、アプリケーションなどから S3 API を呼び出し、SQUID と直接データ通信することが可能となります。データ通信したオブジェクト(ファイル)は、SQUID 内では以下のパスに保存され、フロントエンド環境や計算環境からも直接アクセスが可能です。

```
/sqfs/s3/(UUID)/(バケット名)/(オブジェクト名)
```

7.2.1. アクセスキーの発行

S3 API を利用するためには、アクセスキーを発行する必要があります。アクセスキーの操作はフロントエンド上で s3dskey コマンドにて行います。

アクセスキーの発行は、以下のコマンドを実行します。

```
$ s3dskey create --group=(グループ名)
```

--group : 拡張領域の使用量に含めるグループ名を指定します。

実行に成功すると、次のような出力が返り、アクセスキーが発行されます。

```
+-----------+-----------------------------------------+
| accesskey | AKIA7X1KXXXYYYZZZ000                    |
| enabled   | True                                    |
| fspaths   | None                                    |
| fsuid     | 60101:10                                |
| secretkey | hagwyutos8TThj3SOv9ahPJMF8TTK2dYFcV9Qv7T |
| tag       | (連番):(ユーザ名):(グループ名)                  |
| uuid      | ea60f00a60hufaweofapo12813nfawe9506e1216 |
+-----------+-----------------------------------------+
```

accesskey : S3 API アクセス時に提示するキー

secretkey : アクセスキーに紐づくパスワード情報

tag : アクセスキーを所有するユーザ名、グループ名を含むタグ情報

uuid : キーの一意識別子。ファイルシステムとしてアクセスする際に使用

※ 2021 年 5 月時点で、アクセスキーの発行制限は、1 ユーザ 1 グループあたり、1 本です。

7.2.2. アクセスキーの操作

アクセスキーの操作は、s3dskey コマンドのサブコマンドで行います。サブコマンドの一覧は以下の通りです。

コマンド	説明
s3dskey list	作成済みアクセスキーの一覧表示
s3dskey create	アクセスキーの新規作成
s3dskey disable	アクセスキーの無効化
s3dskey enable	アクセスキーの有効化
s3dskey reset	シークレットキーの初期化

コマンドの詳細は、コマンドヘルプをご参照ください。コマンドヘルプ参照には、以下のコマンドを実行します。

```
$ s3dskey --help
```

7.2.3. バケットの作成

S3 API を通して、オブジェクト(ファイル)をアップロード/ダウンロードするためには、バケットと呼ばれるオブジェクトの管理単位を作成する必要があります。

バケットの作成は、s3dsbucket コマンドにて行います。

```
$ s3dsbucket create --key=(アクセスキー) --bucket=(バケット名)
```

--key　　：事前発行したアクセスキーの指定

--bucket　：作成するバケット名の指定

※ バケット名は、下記の命名規則に従って指定してください。

- 4 文字以上、255 文字未満であること
- 英大文字(A-Z)、英小文字(a-z)、数字(0-9)、ピリオド(.)、ハイフン(-)、アンダースコア(_)のみ指定可能
- バケット名はユニークであり、自身や他ユーザによって作成済みのバケット名と重複できない

※ バケット作成は、S3 API の PUT BUCKET メソッドでも可能です。ただし、フロントエンドでファイルを直接操作した内容が S3 API に同期できないため、s3dsbucket コマンドでの作成を推奨します。

7.2.4. バケットの操作

バケットの操作は、s3dsbucket コマンドのサブコマンドで行います。サブコマンドの一覧は以下の通りです。

コマンド	説明
s3dsbucket list	作成済みバケットの一覧表示
s3dsbucket create	バケットの新規作成
s3dsbucket sync	ファイルシステムのファイルと S3 API のオブジェクトを同期する
s3dsbucket delete	バケットの削除 ※バケット内にオブジェクトが無い場合のみ可能

コマンドの詳細は、コマンドヘルプをご参照ください。コマンドヘルプ参照には、以下のコマンドを実行します。

```
$ s3dsbucket --help
```

7.2.5. API アクセス

オブジェクト(ファイル)操作は、S3 API に対応している各種アプリケーションから行います。以下は、S3 API 利用のため curl のコマンドラッパーである s3curl での利用例です。s3curl についての情報は、以下の URL をご参照ください。

https://github.com/EMCECS/s3curl

(1) アクセス情報の定義

ホームディレクトリ直下に、.s3curl というファイルを作成します。

~/.s3curl

```
%awsSecretAccessKeys = (
        username => {
            id => 'AKIA7X1KXXXYYYZZZ000',
            key => 'hagwyutos8TThj3SOv9ahPJMF8TTK2dYFcV9Qv7T',
        },
);
push(@endpoints, 'squidgw.hpc.cmc.osaka-u.ac.jp')
```

username： アクセス情報のエイリアス。任意の名前を指定可能

id　　： アクセスキー

key　　： シークレットキー

(2) オブジェクトの作成

オブジェクトは、「7.2.3 バケットの作成」で作成したバケットの中にアップロードします。以下のようなコマンドを発行し、オブジェクトをアップロードします。

```
$ LANG=C
$ s3curl.pl --id username --put (ローカルファイル名) -- -s ¥
    -v https://squidgw.hpc.cmc.osaka-u.ac.jp/(バケット名)/(オブジェクト名)
```

アップロードしたオブジェクトは、SQUID フロントエンド上では、下記のパスで参照が可能です。

```
/sqfs/s3/(UUID)/(バケット名)/(オブジェクト名)
```

※ UUID は、「7.2.1 アクセスキーの発行」で表示される、アクセスキーに紐づく識別子です。

7.2.6. バケットの同期

S3 Data Service では、同じオブジェクト(ファイル)に対して、UNIX 的なファイルアクセスと S3 API からのオブジェクトアクセスが可能です。

S3 API 上でのオブジェクト管理(ファイルやディレクトリの存在有無)は、ファイルシステムとは別途管理されているため、フロントエンド等からファイルを直接操作した場合は、変更内容を S3 Data Service へ同期する必要があります。

※ バケットの同期には、s3dsbucket コマンドでバケットを作成する必要があります。

・ コマンドラインから同期する方法

s3dsbucket コマンドの sync サブコマンドを利用します。フロントエンド上で下記の様に実行することで、バケットの同期が可能です。

```
$ s3dsbucket sync --key=(アクセスキー) --bucket=(バケット名)
```

--key　　：事前発行したアクセスキーの指定

--bucket　：同期するバケット名の指定

・ S3 API から同期する方法

S3 Data Service の拡張メソッドである、PUT BucketSync メソッドが利用可能です。s3curl の場合、下記のように実行します。

```
$ LANG=C
$ s3curl.pl --id username --put=emptyPayload -- -s ¥
    -v https://squidgw.hpc.cmc.osaka-u.ac.jp/(バケット名)?sync ¥
    "-H x-ddn-bucket-sync-ops:WRITE,UPDATE,DELETE"
```

エンドポイント名に、「(バケットの URL) + "?syn"」と指定し、必要に応じて、S3 Data Service 用の拡張ヘッダ(x-ddn-bucket-sync-ops)を指定します。引数には、WRITE/UPDATE/DELETE を組

みわせて指定可能です。それぞれの内容は下記の通りです。

- **WRITE**: ファイルシステムをスキャンし、オブジェクトに紐づいていないファイルに対して新しいオブジェクトを紐づける
- **UPDATE**: バケット内のオブジェクトをスキャンし、変更されている場合はオブジェクトのメタデータを更新する
- **DELETE**: バケット内のオブジェクトをスキャンし、紐づくファイルが見つからない場合にオブジェクトを削除する

コマンドラインからの同期や、S3 API からの同期で拡張ヘッダ指定が無い場合は、WRITE,UPDATE,DELETE の指定になります。

7.3. OCTOPUS からのファイル転送

SQUID と OCTOPUS ではファイルシステムが別のシステムになっています。

本項では OCTOPUS も利用している方向けに、SQUID⇔OCTOPUS 間でのファイル転送手順を説明します。

SQUID と OCTOPUS で、同じアカウントを使用している場合のファイル転送方法です。OCTOPUS のファイルシステムは、SQUID のフロントエンドサーバからアクセスすることが可能です。

ファイルシステム	SQUID 上でのファイルパス	OCTOPUS からアクセスする場合のファイルパス
SQUID の高速領域	/sqfs/ssd	/sqfs/ssd
SQUID のホーム領域	/sqfs/home	/sqfs/home
SQUID の拡張領域	/sqfs/work	/sqfs/work

ファイルシステム	OCTOPUS 上でのファイルパス	SQUID からアクセスする場合のファイルパス
OCTOPUS のホーム領域	/octfs/home	/octfs/home
OCTOPUS の拡張領域	/octfs/work	/octfs/work

※SQUID の計算ノードから OCTOPUS のファイルシステムにアクセスすることはできませんので、ファイル転送にのみご利用ください。

以下はファイルの転送方法です。利用者番号を「a61234」としています。

- OCTOPUS → SQUID（SQUID フロントエンド上で作業する場合）
  OCTOPUS のホームディレクトリ下の abc ディレクトリの中のファイル sample.c を、SQUID のホームディレクトリに転送する場合

```
$ cp /octfs/home/a61234/abc/sample.c ~/
```

- SQUID → OCTOPUS（OCTOPUS フロントエンド上で作業する場合）
  SQUID のホームディレクトリ下の abc ディレクトリの中のファイル sample.c を、OCTOPUS のホームディレクトリに転送する場合

```
$ cp /sqfs/home/a61234/abc/sample.c ~/
```

7.4. CIFS でのファイル転送

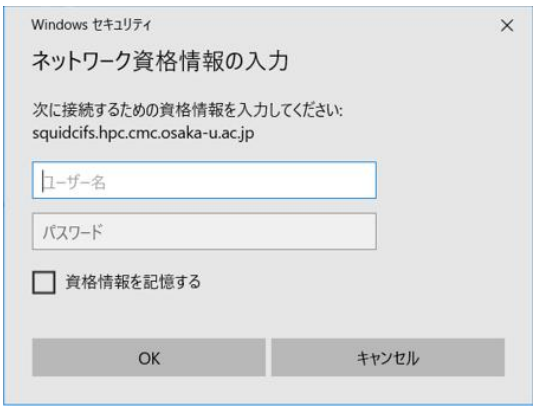
大阪大学のキャンパスネットワークである「大阪大学総合情報通信システム」(ODINS)を使用されている利用者様は SQUID の以下のディレクトリに対して CIFS アクセスが可能です。

- /sqfs/ssd
- /sqfs/work
- /sqfs/home

ご使用の Windows 端末から以下の共有ディレクトリパスで上記ディレクトリにアクセス可能です。

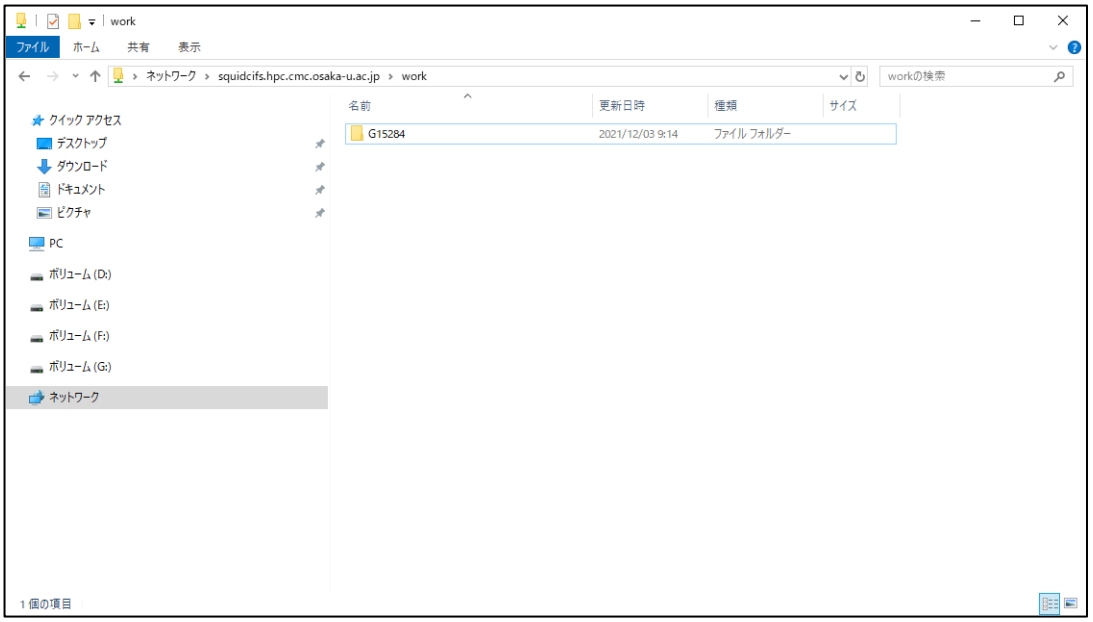
共有ディレクトリパス	アクセス先のディレクトリ
¥¥squidcifs.hpc.cmc.osaka-u.ac.jp¥ssd	/sqfs/ssd
¥¥squidcifs.hpc.cmc.osaka-u.ac.jp¥work	/sqfs/work
¥¥squidcifs.hpc.cmc.osaka-u.ac.jp¥home	/sqfs/home

ユーザ u6b398（所属グループ：G15284）で work ディレクトリにアクセスした実行例を以下に示します。
エクスプローラから、¥¥squidcifs.hpc.cmc.osaka-u.ac.jp¥work でアクセスすると認証ポップアップが表示されます。



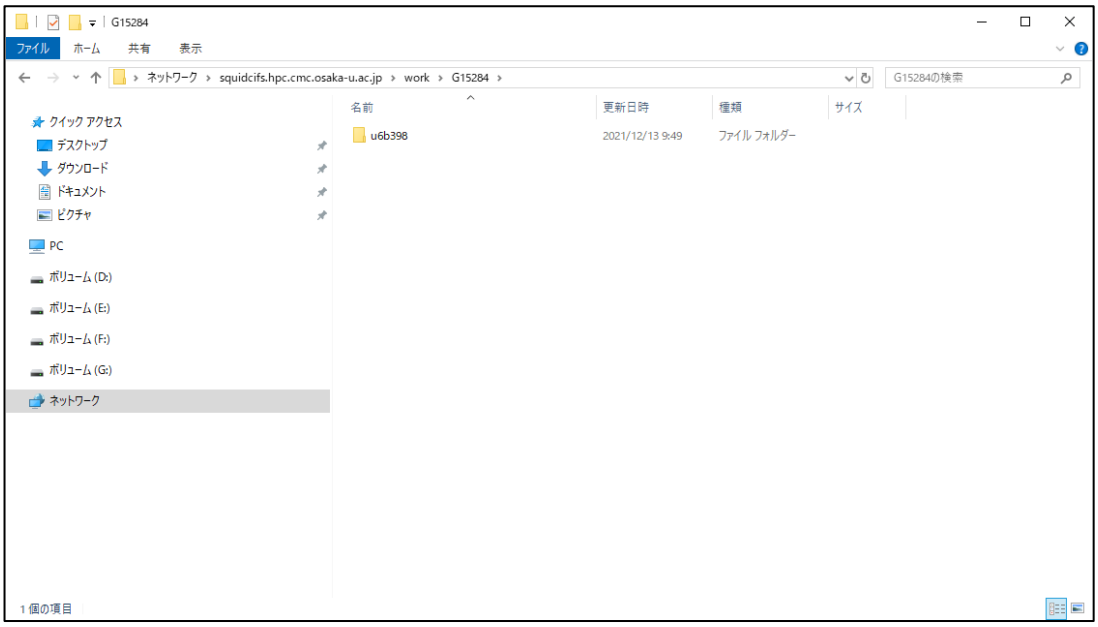
認証ポップアップに SQUID 利用申請して登録されたユーザ名、パスワードを入力すると、work ディレクトリにアクセスすることが可能です。

ssd、work ディレクトリにアクセスすると、最初にご自身が所属するグループのディレクトリが表示されます。



グループディレクトリにアクセスすると、ユーザ専用ディレクトリが表示されます。

ユーザ専用ディレクトリにアクセスし、ファイル転送などにご使用ください。



同様の方法で home ディレクトリにアクセスすると、ユーザ専用ディレクトリが最初に表示されます。

8. その他のサービス

8.1. データ集約用アーカイブストレージ

データ集約用アーカイブストレージとして、オブジェクトストレージ製品である Cloudian 製 HyperStore を利用することができます。

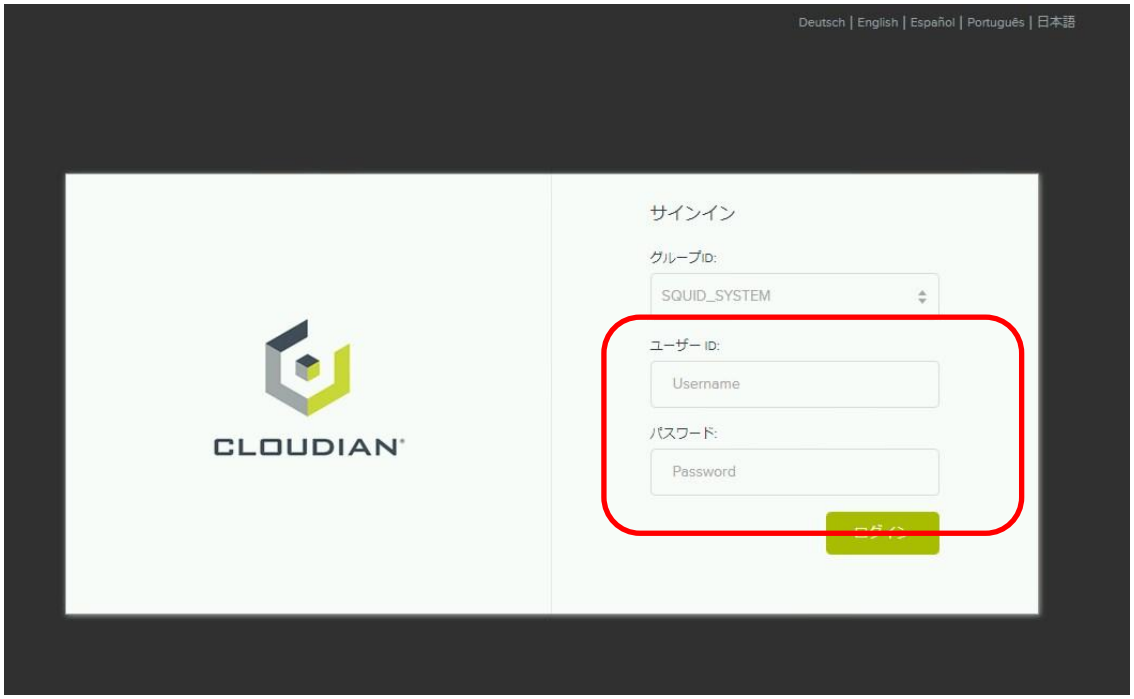
8.1.1. Cloudian の利用方法

(1) ログイン

Web ブラウザを開き、以下の URL を入力します。

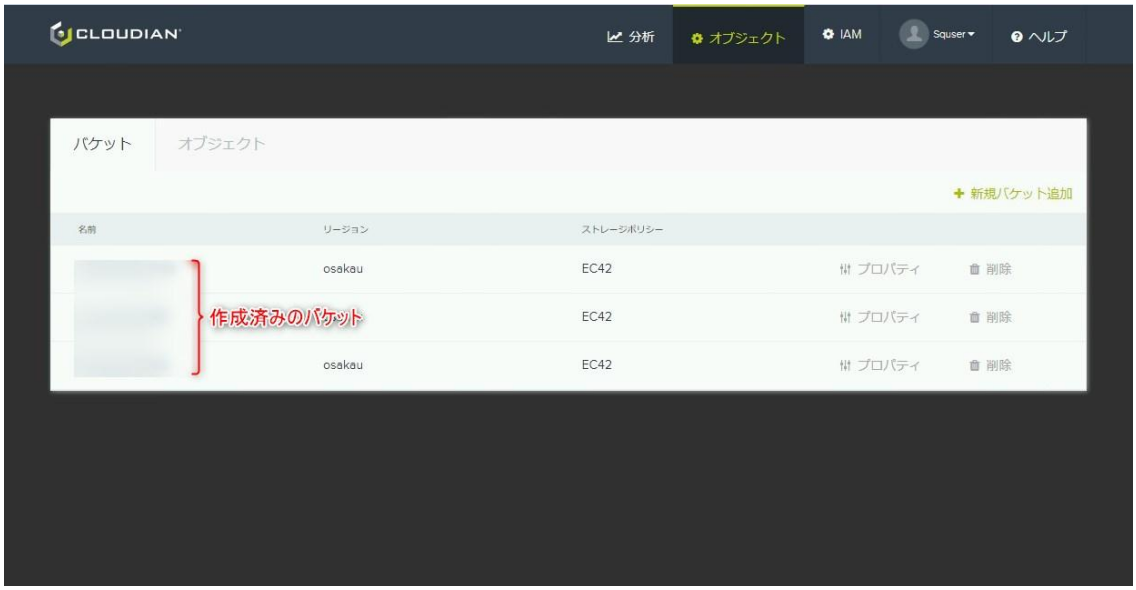
https://onionportal.onion.osaka-u.ac.jp/

以下のログイン画面が表示されますので、ユーザ名とパスワードを入力し、「ログイン」をクリックします。



(2) ログイン後の画面

ログインすると以下の画面になります。

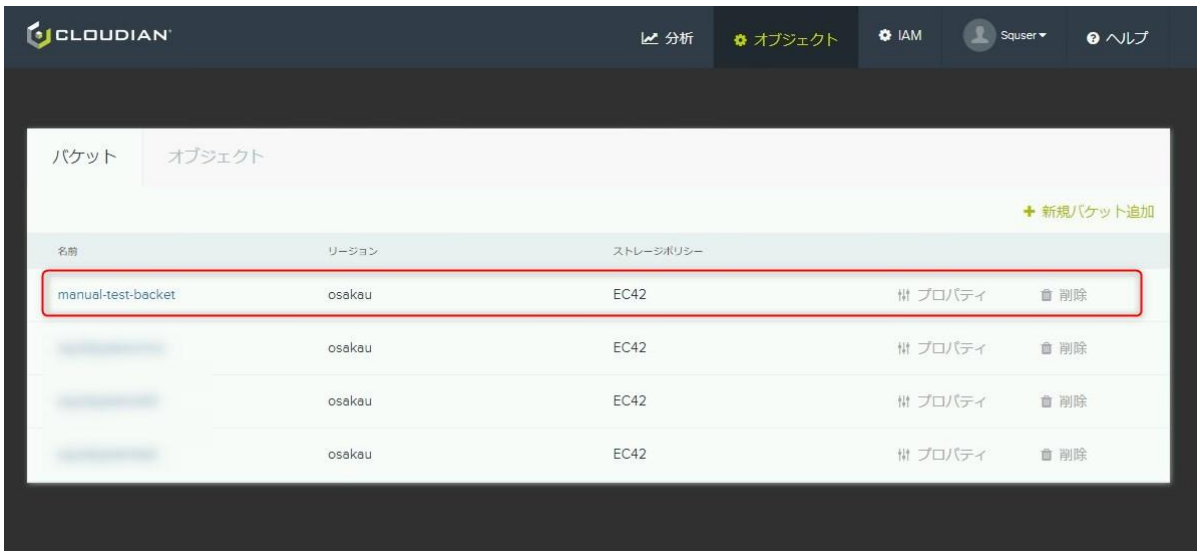


ログイン後、作成済みのバケットが表示されます。

(3) バケット作成

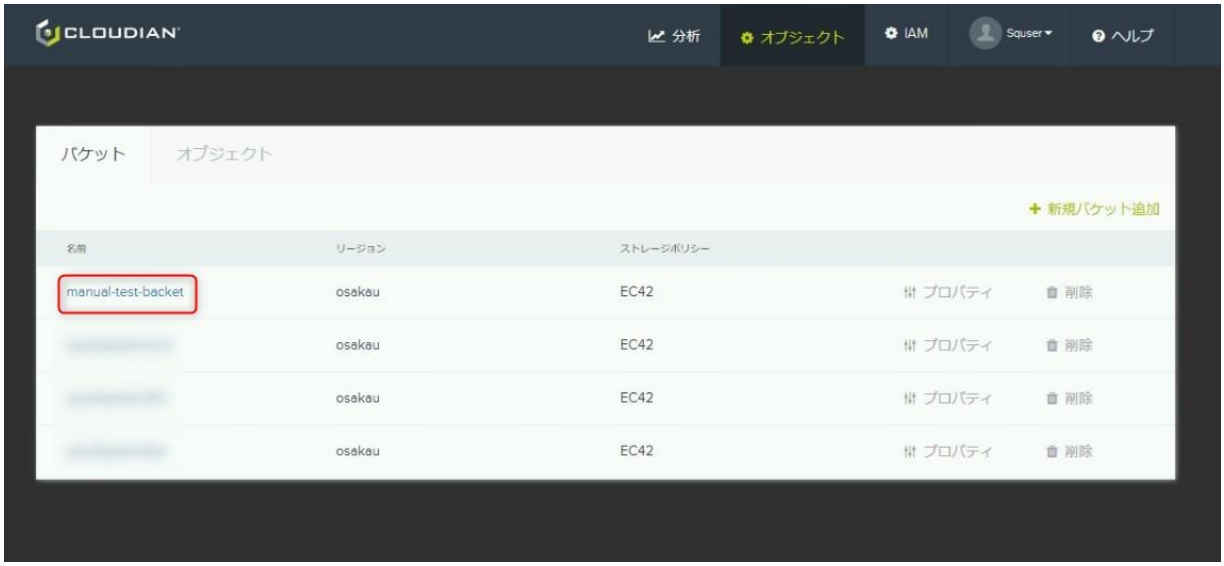
ログイン後の画面から新規バケット追加（①）をクリックし、任意のバケット名を入力します。(②)入力後、作成(③)をクリックします。バケット名に アンダースコアを入力すると機能に制限が発生しますのでご注意ください。

以下のようにバケットが作成されます。



(4) ファイルのアップロード

ログイン後、作成したバケットをクリックし開きます。



ファイルをアップロード をクリックします。

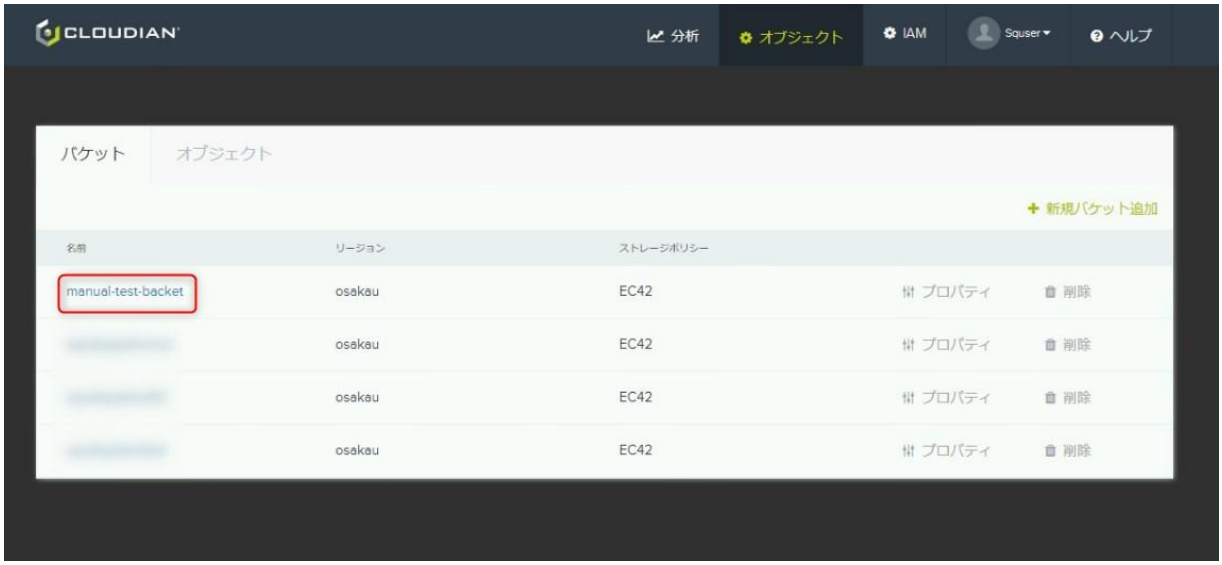
続けて ファイル追加 をクリックします。

アップロードするファイルを選択します。 (本資料では test.txt を選択しています)
アップロード開始 をクリックします。

以下のようにアップロードが完了します。

(5) ファイルのダウンロード

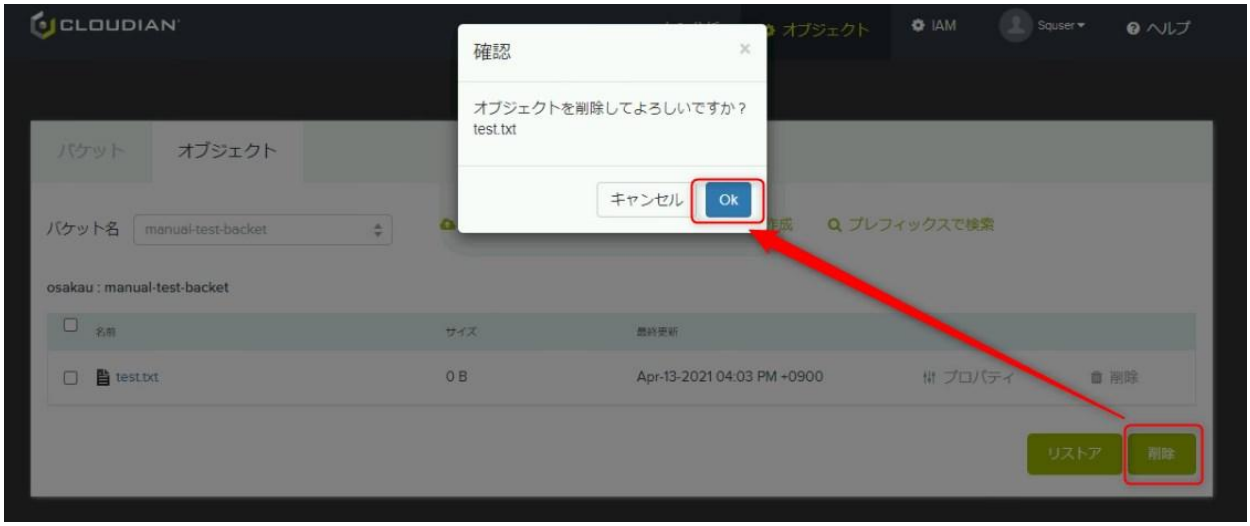
ログイン後、作成したバケットをクリックし開きます。



ファイル名をクリックしダウンロードします。

(6) ファイルの削除

削除するファイルを選択し、 削除 をクリックし確認のうえ OK をクリックします。



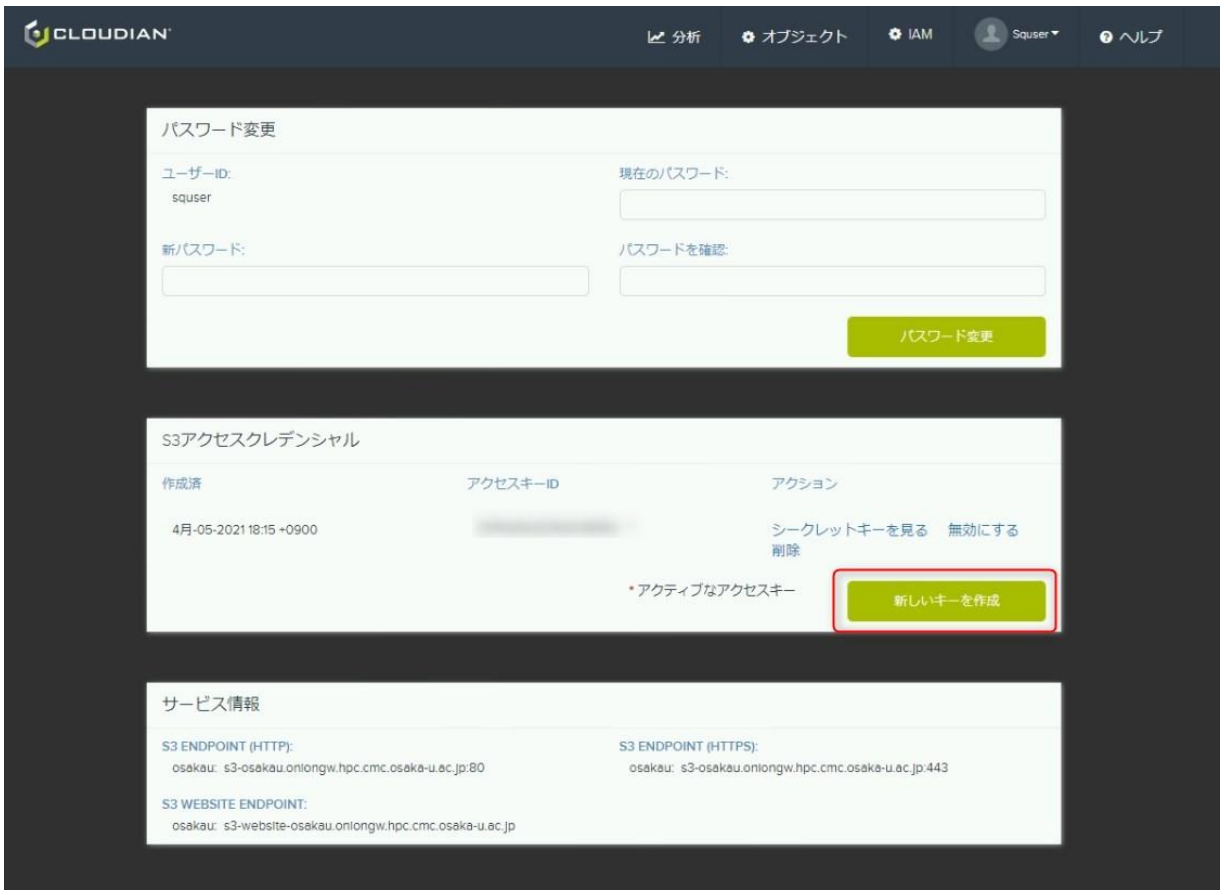
(7) バケットの削除

削除するバケットを選択し、 削除 をクリックし確認のうえ OK をクリックします。

(8) アクセスキーとシークレットキーの発行

ログイン後、ユーザ名　をクリックし 続けて セキュリティ証明書 をクリックします。

S3 アクセスクレデンシャルから 新しいキーを作成 をクリックします



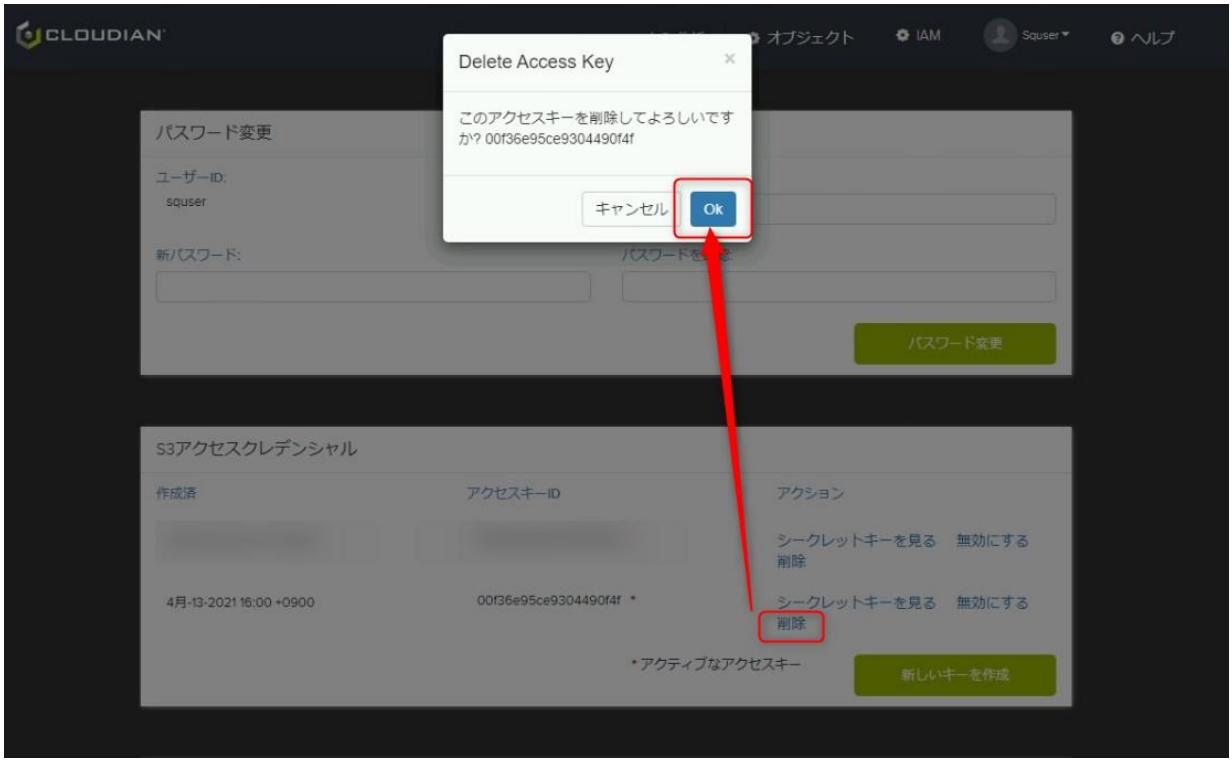
アクセスキー(①)とシークレットキーが発行されます。

シークレットキーは シークレットキーを見る(②) をクリックして表示します。

今後のアクセスに使用するため、アクセスキーとシークレットキーを手元に控えておいてください。

(9) アクセスキーとシークレットキーの削除

S3 アクセスクレデンシャルから削除したいキーの「削除」をクリックし確認の上「OK」をクリックします。

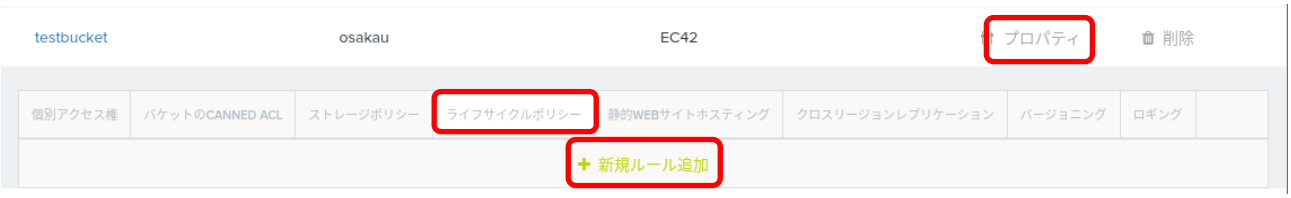


(10) Auto-Tiering

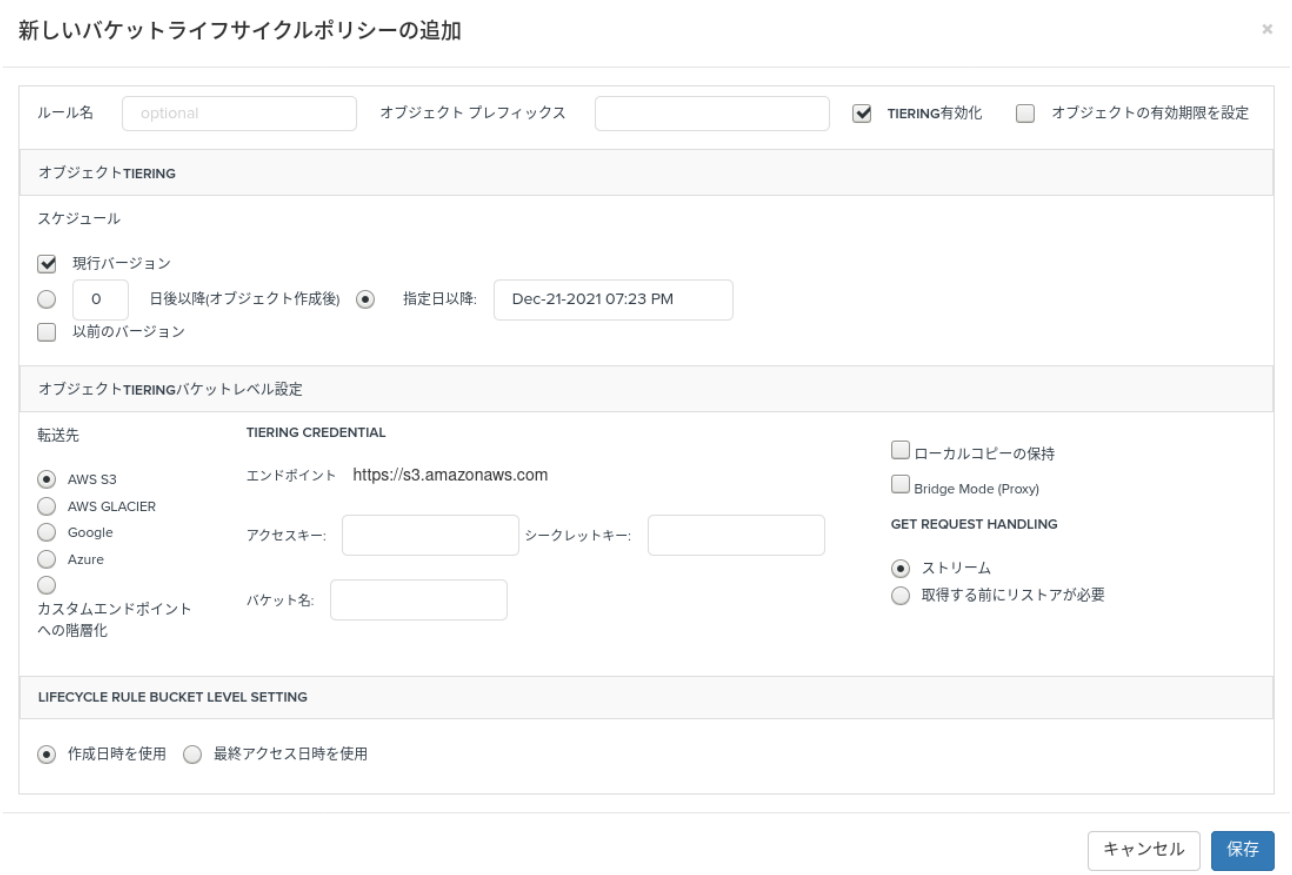
バケット毎に Auto-Tiering 機能で、ライフサイクルポリシーの期間を過ぎたオブジェクトを、他のバケット（Cloudian 以外のベンダーのバケットも可）に自動で移動することが可能です。

Web ブラウザからログインし、Auto-Tiering を設定したいバケットのプロパティをクリックします。プロパティをクリックすると、タブが表示され、その中からライフサイクルポリシーをクリックしてください。

表示される新規ルール追加をクリックしてください。



表示される新しいバケットライフサイクルポリシーの追加画面で TIERING 有効化を選択し、設定を行います。設定内容に問題がある場合は、保存ボタンをクリックすると、エラーメッセージが表示されます。



Auto-Tiering はデフォルトでは 23:00UTC（日本時間 午前８時）に実行されます。

Auto-Tiering によって移動されたオブジェクトは以下の様にオブジェクトのイメージが変化します。

8.2. 統計情報用ポータルシステムの利用方法

統計情報用ポータルシステムでは、以下のサービスを提供しております。

・計算機利用状況 Web 表示

・ノード稼働状況表示

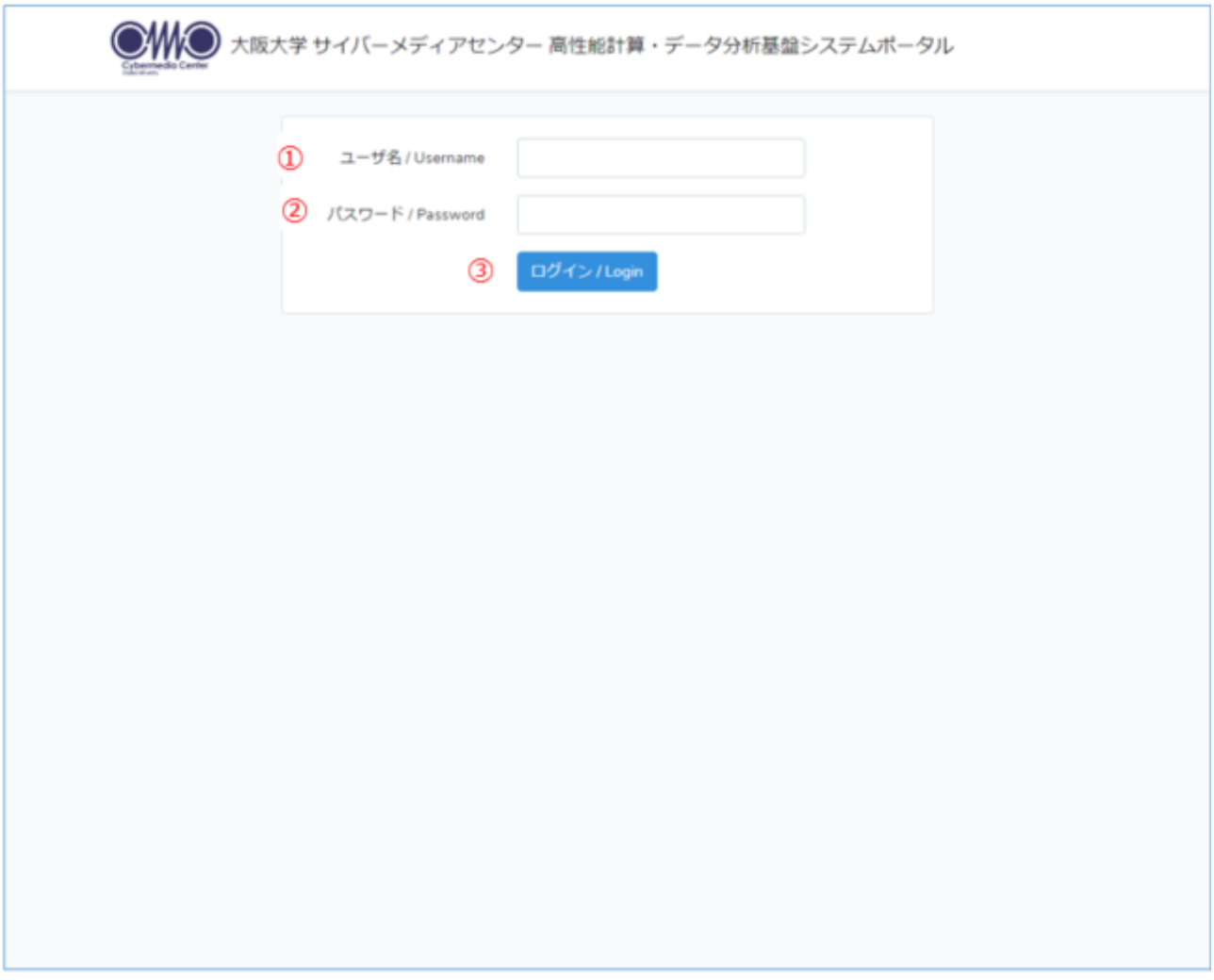
8.2.1. ポータルサイトログイン方法

ポータルサイトへのログインも 2 段階認証の事前準備が必要となります。「2.1.1 2 段階認証アプリのインストール」をご参照の上、事前に 2 段階認証アプリをご用意ください。

※ 認証コードは、フロントエンドログイン用とポータルログイン用で独立していますので、それぞれの初回ログイン時に、2 段階認証設定をお願いします。

(1) ログイン画面

https://squidportal.hpc.cmc.osaka-u.ac.jp/portal/login にアクセスするとポータルシステムのログイン画面が表示されます。



◆ ログイン手順

① 「ユーザ名」に【利用者番号】を入力してください。

② 「パスワード」に【ログインパスワード】を入力してください。

③ 「ログイン」ボタンをクリックしてください。

(2) 2 段階認証設定画面

ログインユーザの 2 段階認証設定が未設定の場合、ログイン認証後、2 段階認証設定画面に遷移します。

◆ 2 段階認証設定手順

① 2 段階認証アプリケーションで表示されている QR コードを読み取ってください。

② QR コードの読み取りができない場合は、シークレットキーを 2 段階認証アプリケーションに入力ししてください。

③ 2 段階認証アプリケーションに表示されている【認証コード】を「認証コード」に入力してく

ださい。

④ 「検証」ボタンをクリックしてください。認証コードの検証処理及び２段階認証設定処理を行います。処理が完了すると、完了ダイアログが表示されます。

◆ 完了ダイアログ

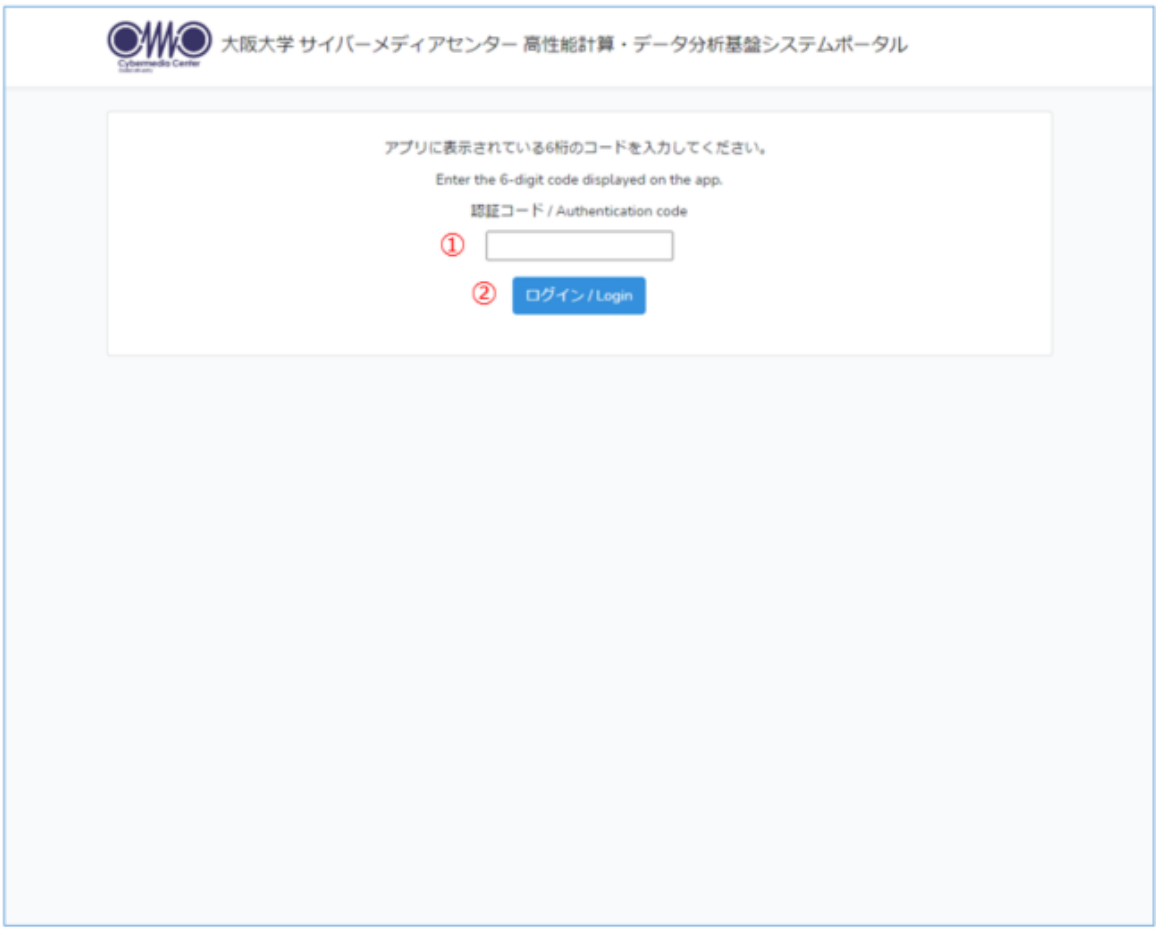
２段階認証設定が完了すると処理完了を示すダイアログが表示されます。

① OK ボタン

「OK」ボタンをクリックすると、ホーム画面に遷移します。

(3) 2 段階認証画面

2 段階認証設定が設定済みの場合、ログイン認証後、2 段階認証画面に遷移します。



◆ 2 段階認証手順

① 2 段階認証アプリケーションに表示されている【認証コード】を「認証コード」に入力してください。

② 「ログイン」ボタンをクリックしてください。

(4) ホーム画面

ホーム画面です。

① ログアウトボタン

「ログアウト」ボタンをクリックすると、ログアウト処理を行いログイン画面に遷移します。

② 計算期利用状況 Web 表示

「状況表示」リンクを選択すると、新しいタブ画面に計算機利用状況を表示します。

③ ノード稼働状況表示

「ノード稼働状況/Node operating status」にある、各 URL を選択することで新しいタブ画面にノード状況を表示します。

汎用 CPU ノードを表示する場合

→ 「汎用 CPU ノード状況表示/General purpose CPU nodes Status display」

GPU ノードを表示する場合

→ 「GPU ノード状況表示/GPU nodes Status display」

ベクトルノードを表示する場合

→ 「ベクトルノード状況表示/Vector nodes Status display」

8.2.2. 計算機利用状況 Web 表示

統計情報用ポータルシステムのホーム画面上で、計算機利用状況の「状況表示」リンクを選択することにより新しいタブ画面に計算機利用状況を表示します。

(1) 利用状況検索画面

利用状況を表示したい年度・期間・グループ・利用者を指定します。

◆ 検索手順

① 年度を指定する

利用状況を表示する年度を選択します。2021 年度～現在の年度までが選択できます。

ただし、利用者登録日以前の年度は選択できません。

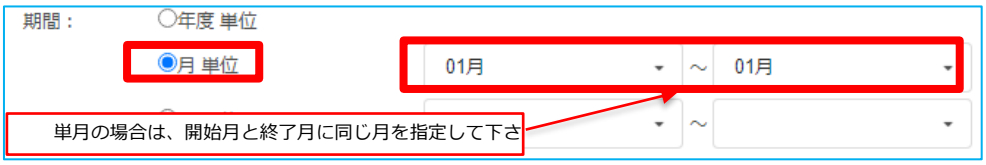
※登録日が 2022/4/1 であれば 2022 年度から選択可能です。

登録日が 2022/3/31 であれば 2021 年度から選択可能です。

② 期間を指定する

・年度単位の合計値を表示する場合は「年度単位」を選択します。

・月単位に表示する場合は「月単位」を選択し、開始月と終了月を入力します。
どちらか一方だけの入力はエラーとなります。



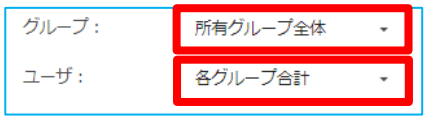
・日単位に表示する場合は「日単位」を選択して、開始日と終了日を入力します。
どちらか一方だけの入力はエラーとなります。

③ グループ・利用者を指定する

ログインユーザの操作権限によって、選択できる対象データが異なります。

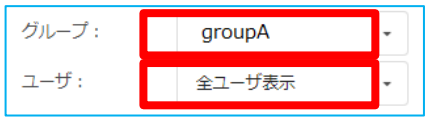
A) 申請代表者

・ 自身が申請代表者を務める全てのグループについて、グループ毎の合計を表示する場合



グループに「所有グループ全体」、
ユーザに「各グループ合計」を指定

・ 自身が申請代表者を務める特定のグループについて、メンバー全員のデータと、グループの合計を表示する場合



任意のグループを指定して、
ユーザに「全ユーザ表示」を指定

・ 特定のグループの特定のメンバーのデータを表示する場合

任意のグループとユーザを指定

B) 利用者

・ 利用者は自身のデータのみ表示できます

自身のグループとユーザ固定。選択できません。

④ 検索

検索ボタンをクリックすることで、検索結果画面を表示します。

※画面が切り替わります。

※検索条件を変更する場合は、ブラウザの戻るボタンをクリックし検索画面を再表示します。

(2) 検索結果画面

指定した条件に応じた情報を閲覧するための画面です。

当画面では閲覧する情報のリンクをクリックすることで、検索結果を表示します。

◆ 利用状況結果の表示手順

対象となる情報のリンクをクリックすると、詳細を別画面で表示します。

※複数のリンクをクリックすることで、それぞれの情報を個別に表示できます。

【対象情報】

A) 計算機の情報

利用ノード時間、投入ジョブ件数

B)　占有/共有毎の情報

利用ノード時間(占有、共有)、投入ジョブ件数(占有、共有)

C)　キュー毎の情報

利用ノード時間、投入ジョブ件数

D)　ファイルシステム

ディスク使用量(ホーム、拡張領域、高速領域)

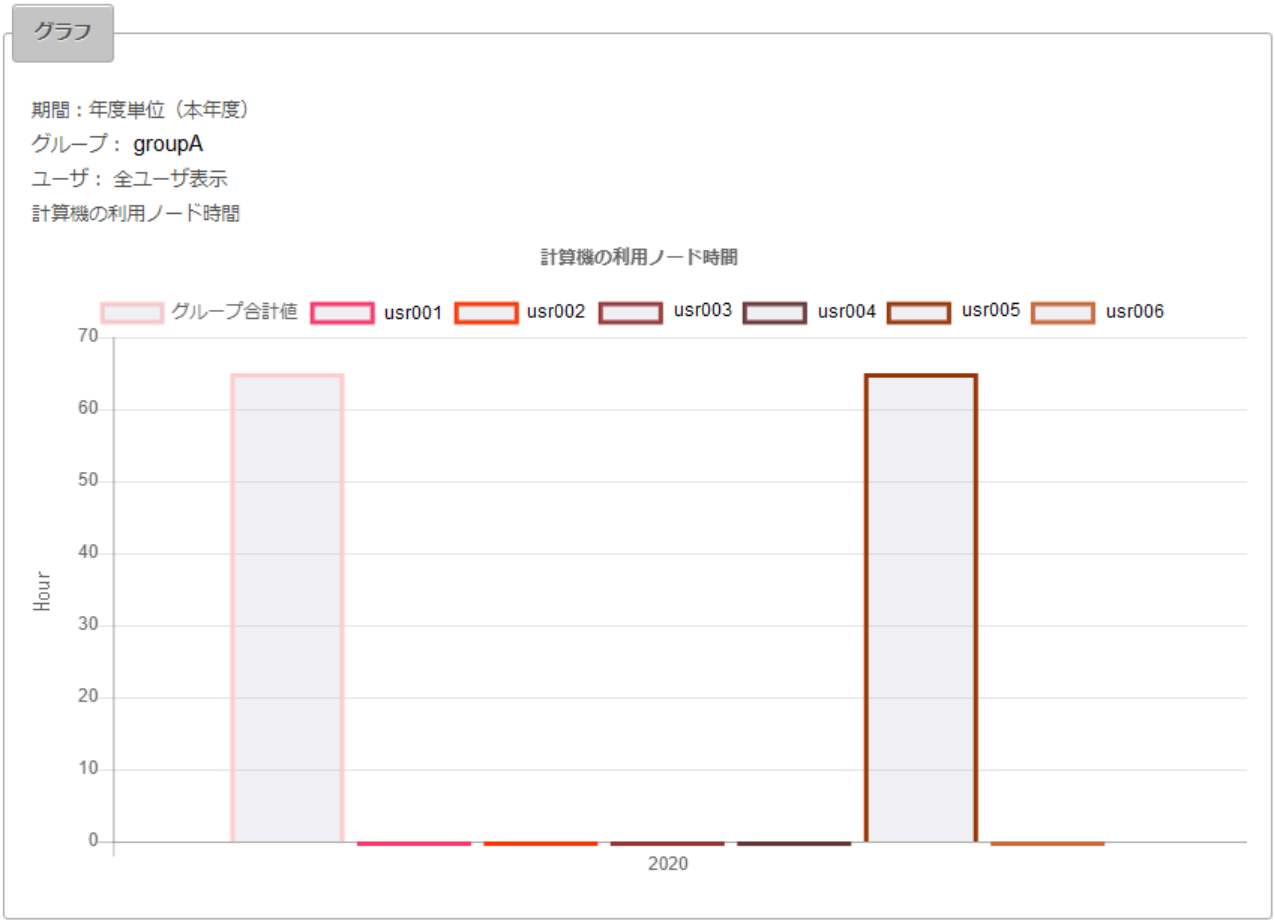
(3) 利用状況結果画面

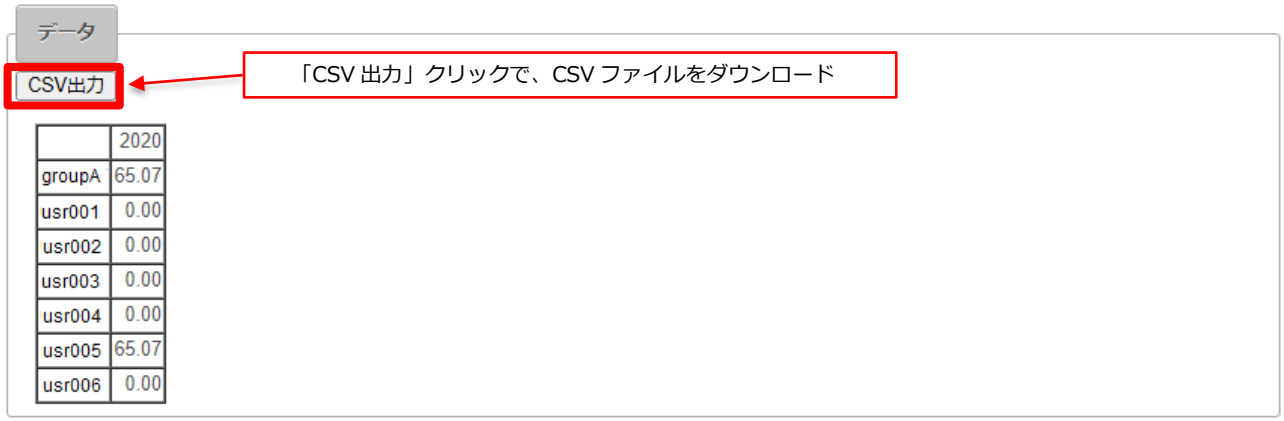
利用状況について、前画面までで指定した条件に基づき、グラフ及びデータを表示します。

※表示データの CSV 出力が可能です。

①　年度単位の条件指定

対象年度の合計値が、グラフおよびデータで表示されます。

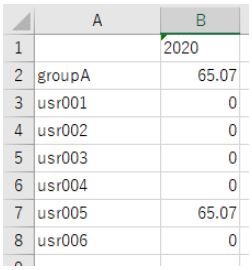




	2020
groupA	65.07
usr001	0.00
usr002	0.00
usr003	0.00
usr004	0.00
usr005	65.07
usr006	0.00

② 年度単位の CSV ファイルイメージ

画面に表示している「データ部」と同じ内容を CSV 形式でダウンロードします。



	A	B
1		2020
2	groupA	65.07
3	usr001	0
4	usr002	0
5	usr003	0
6	usr004	0
7	usr005	65.07
8	usr006	0

※Excel での表示例です

③ 月単位・日単位の条件指定

月(日)毎の推移が、グラフおよびデータで表示されます。

	2020/04	2020/05	2020/06	2020/07	2020/08	2020/09	2020/10	2020/11	2020/12	2021/01	2021/02
usr001	0.00	0.00	0.00	8.00	8.00	8.00	8.00	8.00	8.00	21.00	22.00

④ 月単位・日単位の CSV ファイルイメージ

画面に表示している「データ部」と同じ内容を CSV 形式でダウンロードします。

	A	B	C	D	E	F	G	H	I	J	K	L
1		2020/04	2020/05	2020/06	2020/07	2020/08	2020/09	2020/10	2020/11	2020/12	2021/01	2021/02
2	usr001	0	0	0	8	8	8	8	8	8	21	22

※Excel での表示例です

8.2.3. ノード稼働状況 Web 表示

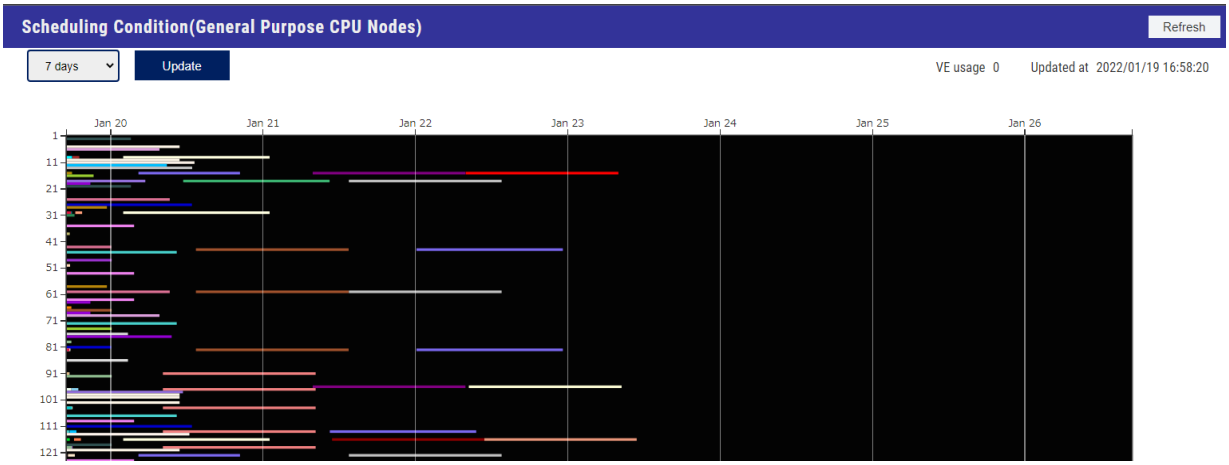
確認を行うノードのリンクを選択します。

以下に汎用 CPU ノードを例に説明します。

「汎用 CPU ノード状況表示/General purpose CPU nodes Status display」に接続後、以下の画面が表示されます。

縦軸はジョブサーバ番号、横軸は時間を示します。

グラフ内の横棒はリクエストを表し、リクエスト毎に色が割り当てられます。



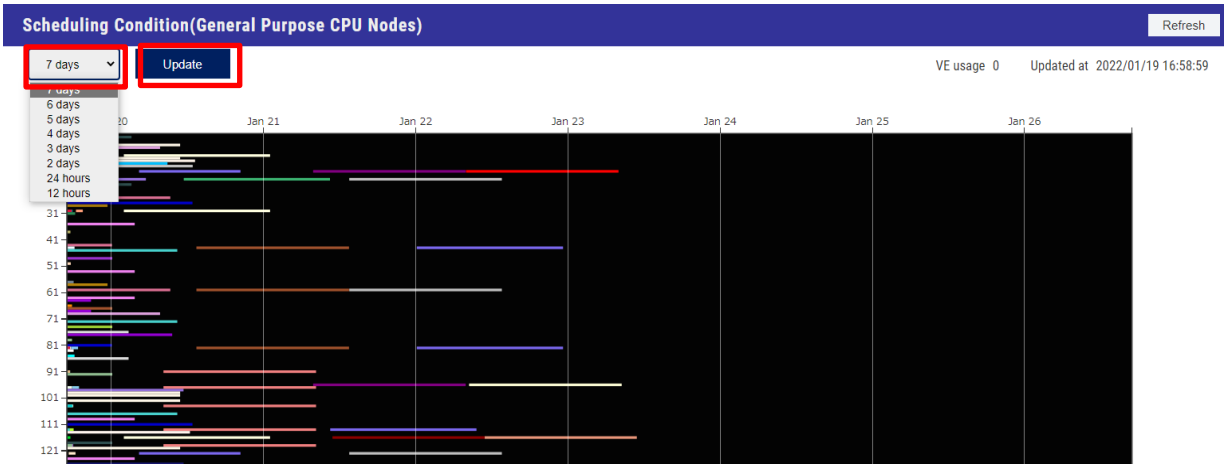
グラフ内の横棒にカーソルを合わせると、リクエスト ID が表示されます。

横棒を押下すると、選択したリクエストのみハイライト状態となります。

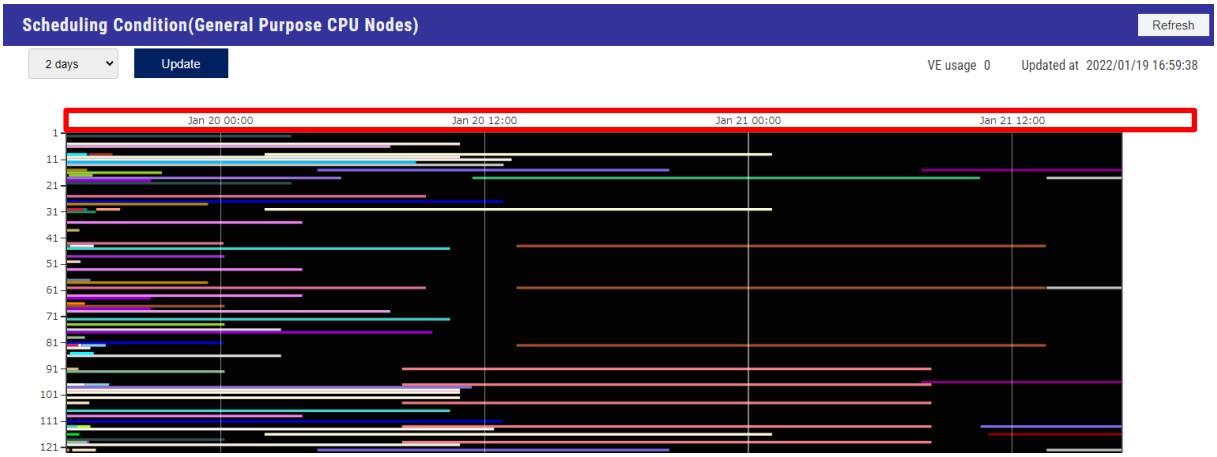
画面右上の「Refresh」ボタンを押下すると、最新のノード状況にページが更新されます。

表示期間を変更する場合は、画面左上のプルダウンから表示したい日数または時間を選択します。

任意の表示期間を選択後、「Update」ボタンを押下します。



表示期間が変更されます。



以上