発行番号：OUD3C-share-技術-003-08

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

目次

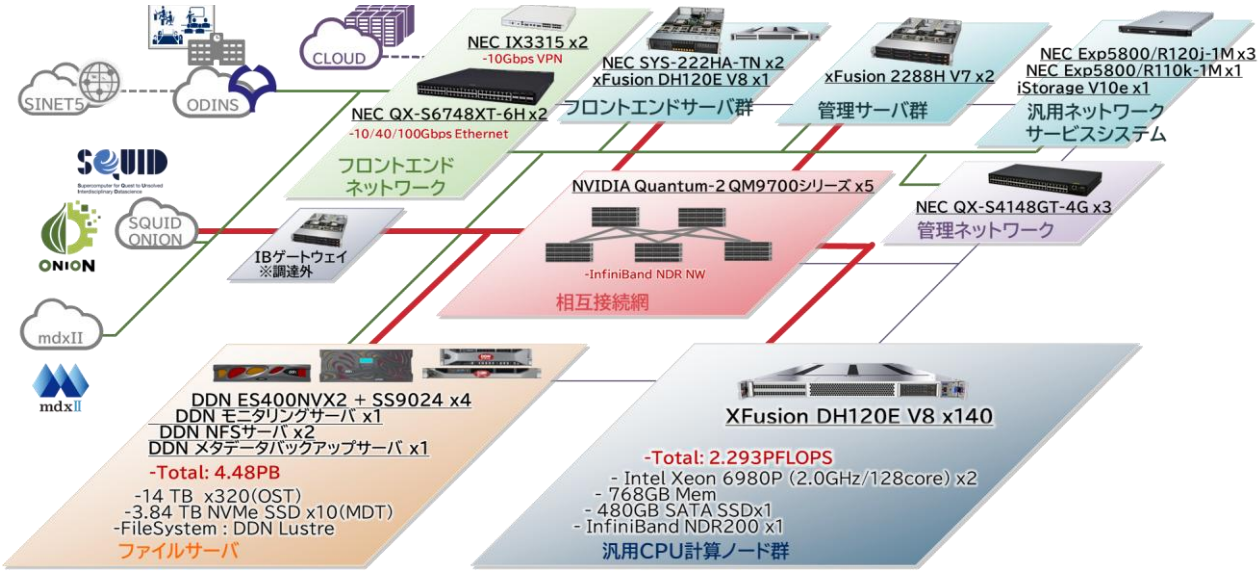
改版履歴

版数	年月日	該当項目	概要説明	承認	査閲	作成
1.0	2025/08/29	全項目	新規作成	NEC	NEC	NEC
2.0	2025/09/02	6.1 6.3.10	MotionCor3 の実行方法修正	塩田	塩田	大塚
3.0	2025/09/11	5.1.2	qlogin のオプション修正	塩田	塩田	西山
4.0	2025/09/22	6.1 6.3.7 6.3.17 6.3.18	アプリケーション一覧に「SMASH」と「NEC Vector Annealing」追記 GENESIS の追記 SMASH の追記 NEC Vector Annealing の追記	山本	塩田	北野
5.0	2025/11/27	3.1.2	利用状況の確認方法の追記	塩田	塩田	大塚
7.0	2026/01/14	5.1.4	スケジューリング状態を追記 以下のコマンドを追記 sstat のオプション有無の挙動 sstatgroup sstatall	塩田	塩田	大塚
8.0	2026/04/10	5.1.5	NQSV における Core ファイル出力設定を追記	塩田	塩田	大塚

1 システム概要

1.1 システム全体図

「OCTOPUS」は、汎用 CPU 計算ノード群、ファイルサーバから構成され、総理論演算性能 2.293 PFLOPS を有するスーパーコンピュータシステムです。



ノード構成	汎用 CPU 計算ノード群 140 ノード（2.293 PFLOPS）	演算機：Intel(R) Xeon(R) 6980P (2.0GHz/128core) x2 メモリ容量：768GB
相互接続網	InfiniBand NDR200 (200 Gbps)	
ファイルサーバ	DDN EXAScaler (HDD 4.48PB)	

1.2 計算環境

OCTOPUS では、計算環境として汎用 CPU 計算環境が利用可能です。

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、「DH120E V8」140 台で構成されています。

DH120E V8

・ 汎用 CPU 計算ノード

項目		構成
総ノード数		140 台
サーバ構成	プロセッサ	Intel(R) Xeon(R) 6980P (2.0GHz/128core) x2
	メモリ構成	768 GB (32GB DDR5-6400 DIMM x24)
	ハードディスク	480GB SATA SSD x1
	インタフェース	InfiniBand NDR200 x1, 1000BASE-T x2

・ ソフトウェア環境

項目	構成
OS	Rocky Linux 9.4 (64bit)
コンパイラ	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++/FORTRAN の開発環境である「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。

1.3 ストレージ領域

OCTOPUS では、以下のストレージ環境が利用可能です。

ストレージ領域	HDD ストレージ
特徴	高速かつ大容量の データ格納領域
容量	home：100GiB work：5 TiB+追加購入
ファイルシステム	DDN ExaScaler (Lustre)
有効利用容量	3.58PB
ディスク装置	14TB 7,200rpm ML-SAS-HDD
ハードウェア	DDN ES400NVX2 + SS9024 x4

OCTOPUS では、データ利活用工場を目的として、さまざまなデータアクセス方法を提供します。
各アクセス方法の参照先を以下に示します。

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	2.3 ファイル転送

		データ転送方法	
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
システム間アクセス	NFS	周辺システム(SQUID)とのデータ交換方法	7.2 SQUID からのファイル転送

1.4 2種のフロントエンド

1.4.1 各フロントエンドの特徴

OCTOPUS では、2 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	占有型フロントエンド
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	仮想マシンによる利用グループ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	2	(個別申請により増減)
アプリケーション	Amazon DCV Server	
その他	NVIDIA L4 x1	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、「SYS-222HA-TN」2 ノードで構成されます。CPU として「Intel(R) Xeon(R) 6980P」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		2 ノード
サーバ構成	プロセッサ	Intel(R) Xeon(R) 6980P (2.0GHz/128core) x2
	メモリ構成	768 GB (32GB DDR5-6400 DIMM x24)

	ハードディスク	960GB SATA SSD x2
	インタフェース	InfiniBand NDR200 x1, 10GBASE-T x2, 1000BASE-T x2, BMC x1
	GPU	NVIDIA L4 x1

・ ソフトウェア環境

項目	構成
OS	RHEL 9.4 (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	Amazon 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「Amazon 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」参照ください。

1.4.3 占有型フロントエンド

占有型フロントエンドが稼働する環境は 1 ノードで構成されます。CPU として「Intel(R) Xeon(R) 6980P」を有します。別途利用申請を行うことで、仮想マシンによる申請グループ専用のフロントエンド環境を用意します。

・ 占有型フロントエンド

項目		構成
総ノード数		1 ノード
サーバ構成	プロセッサ	Intel(R) Xeon(R) 6980P (2.0GHz/128core) x2
	メモリ構成	1,152 GB (48GB DDR5-6400 DIMM x24)
	ハードディスク	960GB SATA SSD x2
	インタフェース	InfiniBand NDR200 x1, 10GBASE-T x2, 1000BASE-T x2, BMC x1

・ ソフトウェア環境

項目	構成
OS	RHEL 9.4 (64bit)
ジョブ投入環境	NEC NQSV

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。
ジョブ投入環境の利用方法は「5 プログラム実行方法」をご参照ください。

1.5 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
	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

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。

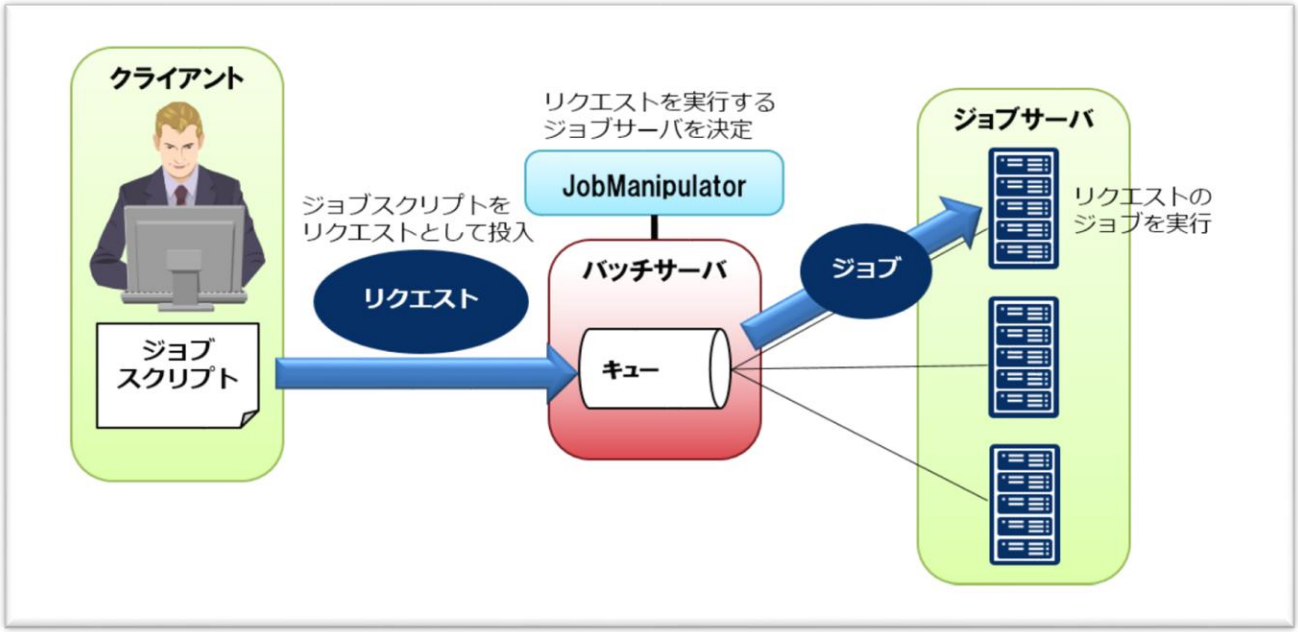
- Environment Modules 公式ページ
  http://modules.sourceforge.net/

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、OCTOPUS 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。

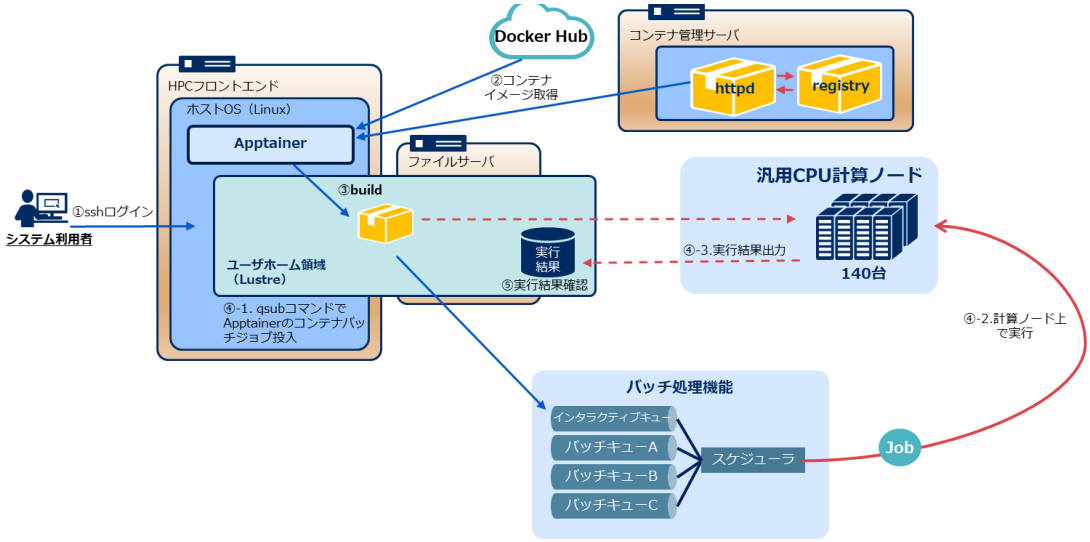
OCTOPUS では、多人数のユーザで共有利用されることが前提となっているため、このようなバッチ処理方式を主なプログラム実行方針として採用しています。



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、Apptainer 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、「4.4 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、「5.4 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

Apptainer 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「4.4.1 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
- OCTOPUS ローカルレジストリからイメージを取得する
- Apptainer 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

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「4.4.2 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。「5.4 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。

Apptainer 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。

- Apptainer 公式ページ
  https://apptainer.org/

- Apptainer 1.4 User Guide
  https://apptainer.org/docs/user/1.4/

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

OCTOPUS にアクセスするためには、全国共同利用スーパーコンピューティング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 D3C-Sinet にアクセスする必要があります。本システムでは OCTOPUS へのアクセスのために、HPC フロントエンド、占有型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	octopus.hpc.osaka-u.ac.jp
2	占有型フロントエンド	voctopusXX.hpc.osaka-u.ac.jp

※占有型フロントエンドを使用する場合は、個別申請が必要です。申請が承認されると接続先が連絡されます。

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。

接続確認済みの 2 段階認証アプリケーションは以下の通りです。

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**

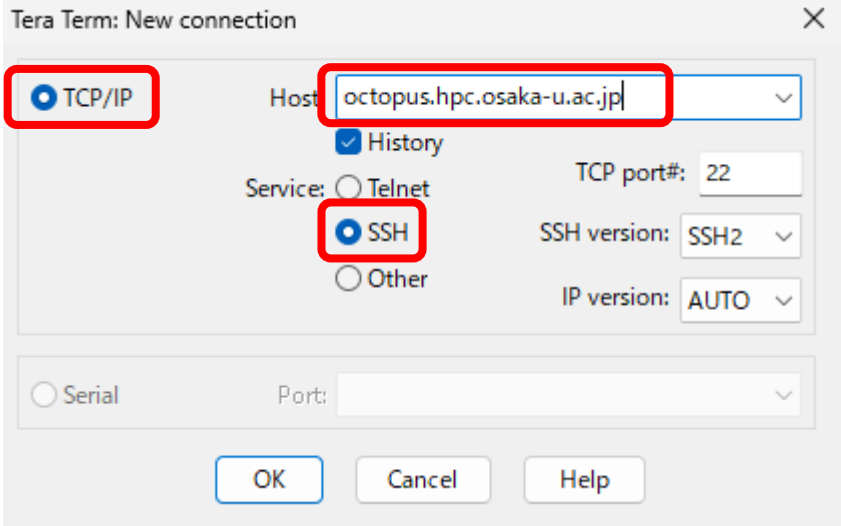
ssh コマンドを使用します。
HPC フロントエンド(octopus.hpc.cmc.osaka-u.ac.jp)にログインする例は以下の通りです。

```
$ ssh (-l ユーザ名) octopus.hpc.osaka-u.ac.jp
The authenticity of host 'octopus.hpc.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)@octopus.hpc.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。)
```

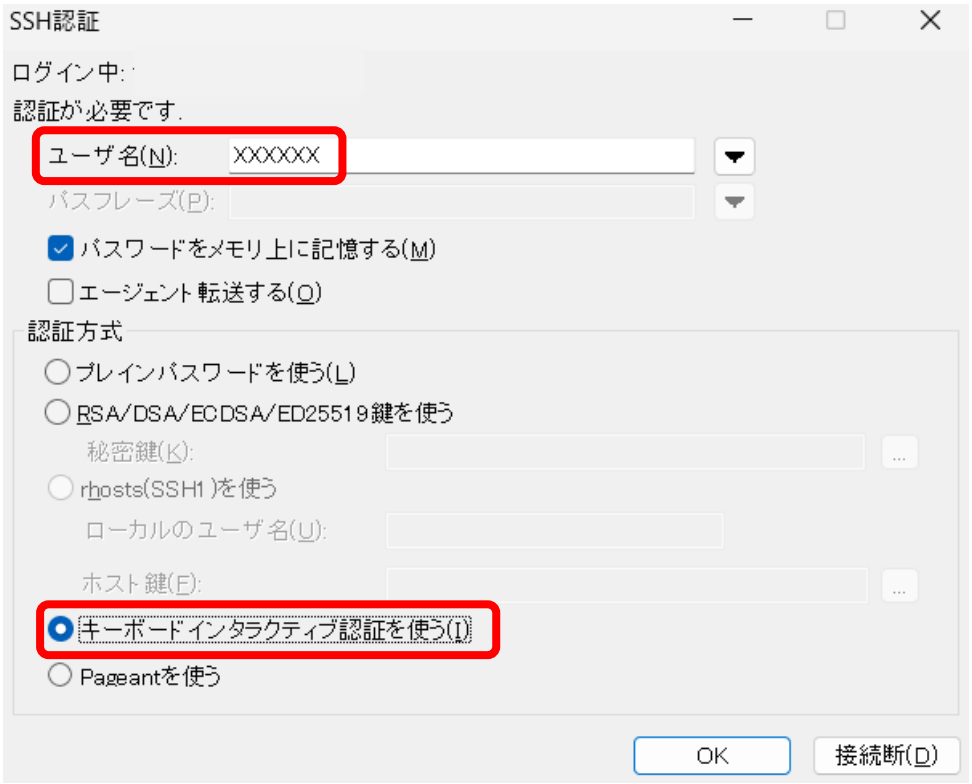
※(-l ユーザ名) : ログインユーザ名がローカルマシンのユーザ名と異なる場合に指定

- **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 octopus.hpc.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。

※ターミナルのウィンドウが小さいと QR コードの表示が崩れます。ウィンドウサイズを最大にして接続することを推奨します。

```
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@octop
us.hpc.osakau.ac.jp%3Fsecret%3DDXXXXXXXXCLI%26issuer%3Doctopus.hpc.osaka-u.ac.jp
```

Your new secret key is: XXXXXXXXXXX

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。

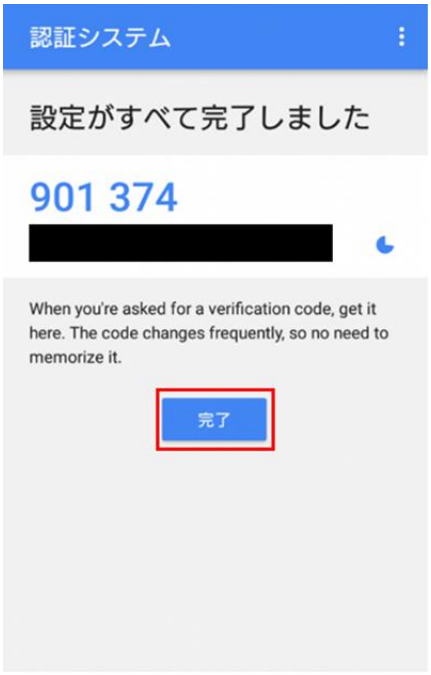
(3) 2 段階認証アプリの設定

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)
```

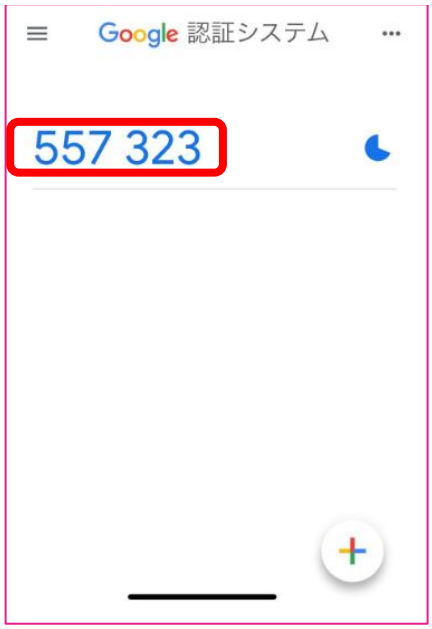
「Enter code from app (-1 to skip):」に対しては**-1 を入力しスキップしてください。**
「ok? Hit enter:」が表示されます。Enter キーを押すと 1 度ログアウトします。

※「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、OCTOPUS 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。

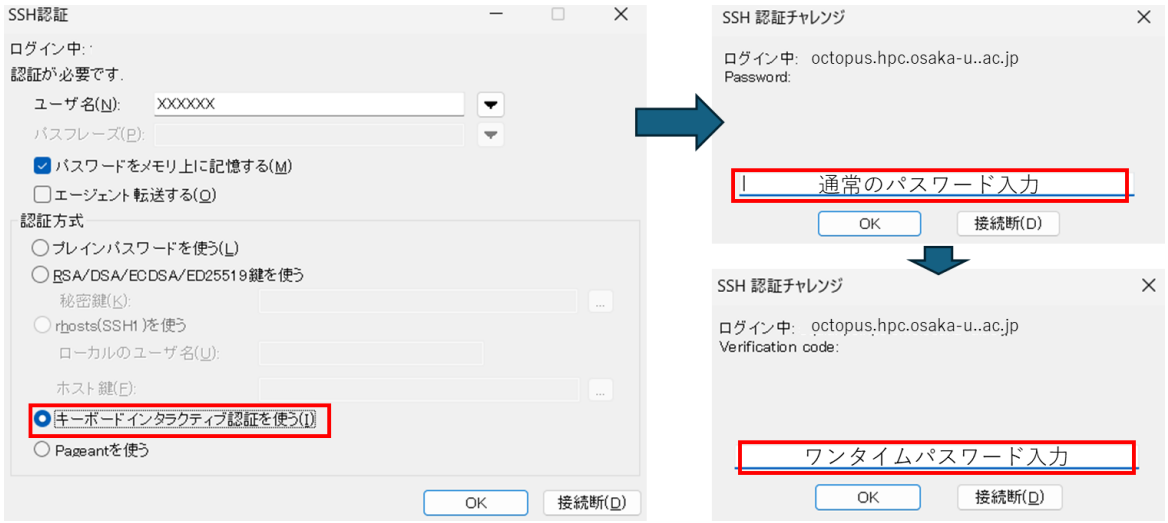
- **UNIX 系 OS(Linux、macOS)からのログイン**
  ssh コマンドを使います。フロンエンド(octopus.hpc.osaka-u.ac.jp)に SSH ログインする例は以下の通りです。

```
$ ssh (-l ユーザ名) octopus.hpc.osaka-u.ac.jp
(ユーザ名)@octopus.hpc.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 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 OAuth でのログイン方法

HPCI 利用者の場合、OAuth 認証によるログインが可能です。

ログイン方法は、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 OAuth 対応版』を参照ください。

- HPCI システム利用情報・マニュアル等
  https://www.hpci-office.jp/for_users/hpci_info_manuals
  - HPCI ログインマニュアル OAuth 対応版(HPCI-CA01-003-02 ※2025 年 8 月時点)

HPC フロントエンドサーバ(octopus.hpc.osaka-u.ac.jp)が OAuth でログイン可能です。

2.3 ファイル転送

2.3.1 UNIX 系 OS（Linux, macOS）でのファイル転送

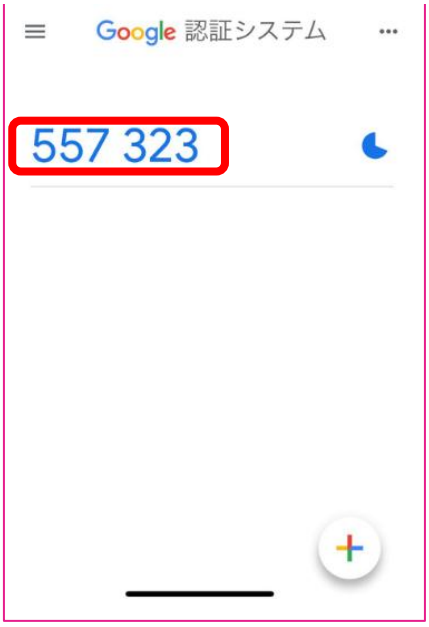
ここでは、D3 センター外部の研究室などの UNIX 系 OS（Linux, macOS）端末から、FTP サーバへ接続する例を紹介します。端末側に予め SFTP や SCP がインストールされていることを前提としています。SFTP や SCP がインストールされていない場合は、ご自分でインストールしていただくか、端末の管理者にご相

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@octopus.hpc.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 'octopus.hpc.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.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@octopus.hpc.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 OCTOPUS のホームディレクトリに転送する例です。

```
sftp> put sample.f
Uploading sample.f to /octfs/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、OCTOPUS のホームディレクトリに転送する場合

```
$ scp sample.f a61234@octopus.hpc.osaka-u.ac.jp:
```

・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、 ローカル端末のカレントディレクトリ上に sample2.c として転送する場合

```
$ scp a61234@octopus.hpc.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

始めて 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 'octopus.hpc.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.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@octopus.hpc.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。

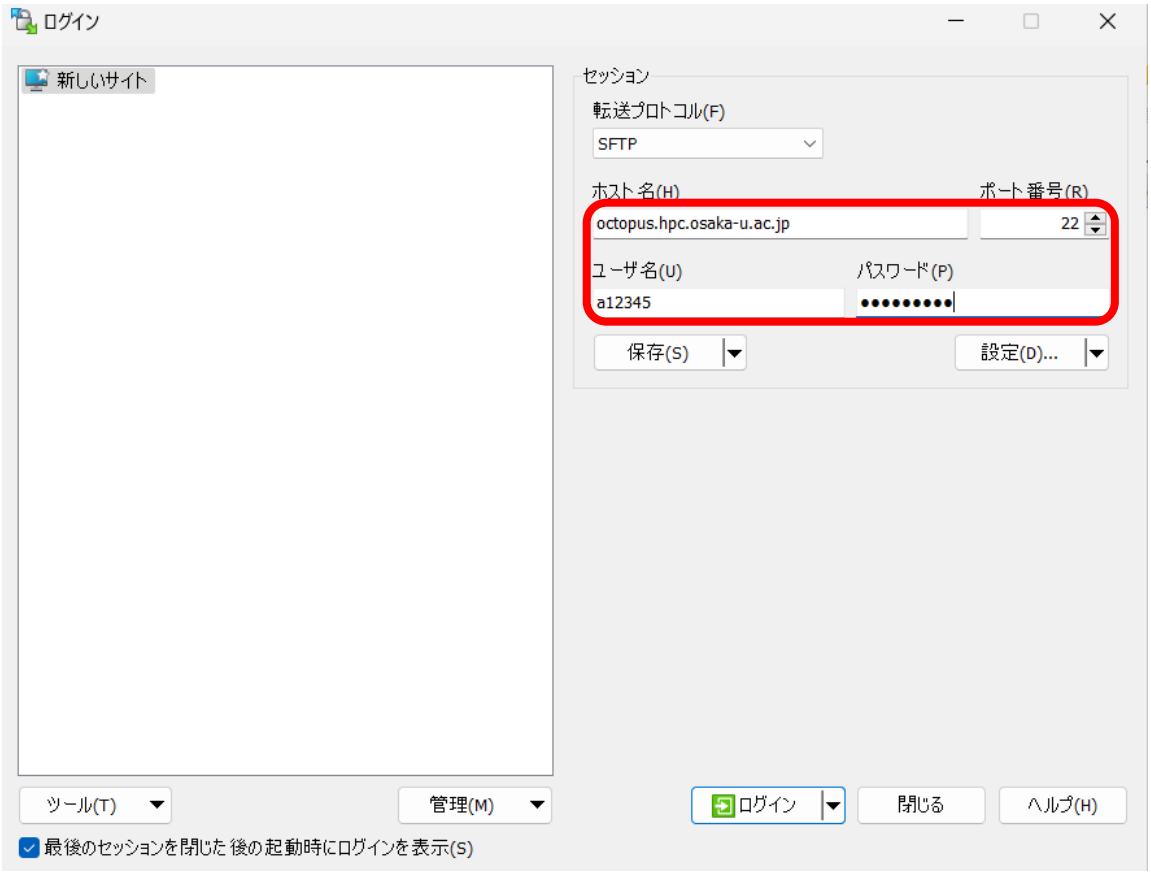
(1) WinSCP の入手

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

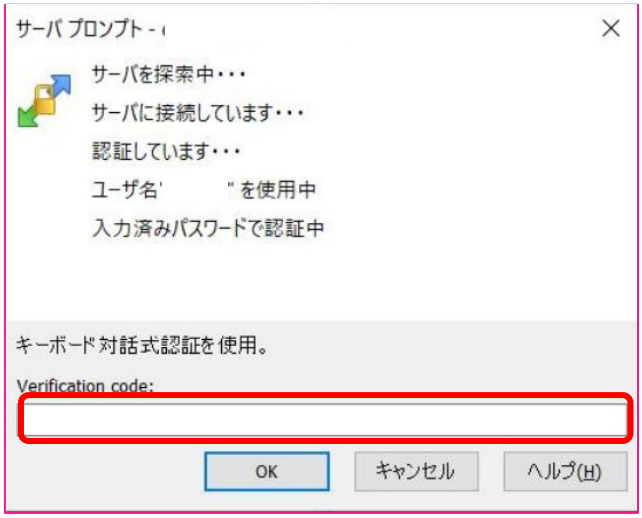
WinSCP を起動し、FTP サーバへ接続します。以下の項目を設定後、[ログイン]ボタンをクリックします。

```
ホスト名:  octopus.hpc.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

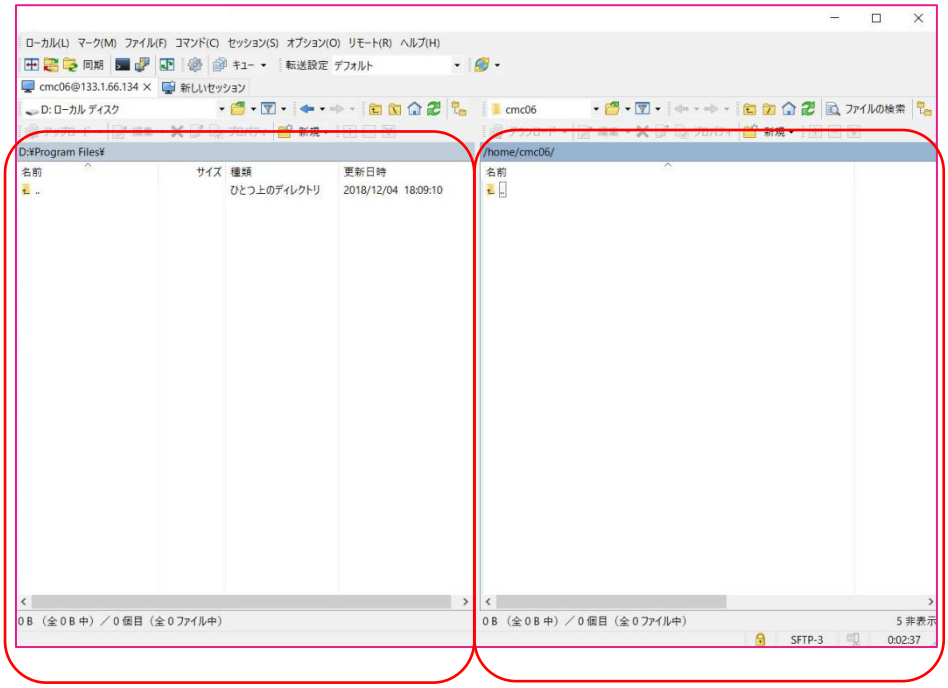
Verification code を聞かれますので、2 段階認証アプリから取得したワンタイムパスワードを入力します。



※プログラムソースや NQSV のジョブ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(OCTOPUS)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(OCTOPUS)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 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	/octfs/home/(利用者番号)	100GiB	---	ブラウザアクセス領域を含む
		拡張領域	/octfs/work/ (グループ名)/(利用者番号)	5TiB (+)	Soft:3 億 Hard:5 億	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、100GiB が割り当てられます。
  ファイルパス(home)は以下になります。

  ```
  /octfs/home/(利用者番号)
  ```

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

- **拡張領域**

  高速かつ大容量の I/O が可能な領域です。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。
  ファイルパス(work)は以下になります。

  ```
  /octfs/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　/octfs/work/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
```

上記で表示される項目は以下の通りです。

[Group summary]

- OCTOPUS 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、申請したボーナスポイント、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。

- 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。

[Detail]

- OCTOPUS 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。

- Bonus points

ユーザ毎に、ユーザが現在までに使用したボーナスポイント、申請したボーナスポイント、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。

- 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。

[Node-hours]

- Usage

グループが現在までに使用したノード時間を表示します。

- Usage(Bonus)

グループが現在までにボーナスで使用したノード時間を表示します。

- Available

使用可能な残り OCTOPUS 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 OCTOPUS 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 OCTOPUS 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 OCTOPUS 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
	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/2025   BaseCPU/2025   BaseExtra/2025   BaseGCC/2025   BaseJDK/2025   BaseJulia/2025   BasePy/2025
BaseR/2025
```

② モジュールのロード

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

③ ロードされているモジュールの確認

```
$ module list
Currently Loaded Modulefiles:
 1) tbb/latest            3) umf/latest         5) mpi/latest        7) dal/latest                  9)
mkl/latest    11) debugger/latest  13) inteloneAPI/2025.2.0
 2) compiler-rt/latest    4) compiler/latest    6) advisor/latest    8) intel_ipp_intel64/latest   10)
vtune/latest  12) dpl/latest       14) BaseCPU/2025
```

④ 追加でロード可能なモジュール一覧の表示

```
$ module avail
-------------- /system/apps/rhel9/cpu/InteloneAPI/InteloneAPI2025.2/2025.2.0/modulefiles --------------
advisor/2025.2   compiler-intel-llvm/2025.2.0   compiler/2025.2.0   debugger/2025.2.0          dnnl/3.8.1
dpl/2022.9                intel_ippcp_intel64/2025.2  mkl/2025.2   tbb/2022.2  vtune/2025.4
advisor/latest   compiler-intel-llvm/latest     compiler/latest     debugger/latest            dnnl/latest
dpl/latest                intel_ippcp_intel64/latest  mkl/latest   tbb/latest  vtune/latest
```

```
ccl/2021.16.0    compiler-rt/2025.2.0          dal/2025.6          dev-utilities/2025.2.0  dpct/2025.2.0
intel_ipp_intel64/2022.2  ishmem/1.3.0              mpi/2021.16  umf/0.11.0
ccl/latest       compiler-rt/latest            dal/latest          dev-utilities/latest    dpct/latest
intel_ipp_intel64/latest  ishmem/latest             mpi/latest   umf/latest

-------------- /system/apps/env/cpu/lib/inteloneAPI2025.2 --------------
hdf5/1.14.6    netcdf-c/4.9.3    netcdf-cxx/4.3.1    netcdf-fortran/4.6.2    pnetcdf-c/1.14.0    pnetcdf-
cxx/1.14.0  pnetcdf-fortran/1.14.0

-------------- /system/apps/env/cpu/Compiler --------------
inteloneAPI/2025.2.0

------------------ /system/apps/env/Base --------------
BaseApp/2025   BaseCPU/2025   BaseExtra/2025   BaseGCC/2025   BaseJDK/2025   BaseJulia/2025   BasePy/2025
BaseR/2025
```

(5) 追加モジュールのロード

```
$ module load hdf5/1.14.6
```

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 v5.3.0 : Docs >> module
  https://modules.readthedocs.io/en/v5.3.0/module.html

3.1.4 ulimit 設定

フロントエンドでは、ulimit 設定により Core ファイルの出力制限を実施しています。既定値は Core ファイルの出力サイズ（Soft）が 0 に設定されています。設定を行わない場合、Core ファイルは出力されません。フロントエンド上で Core ファイルを出力させる場合は、利用者様自身で ulimit 設定を行う必要があります。

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。。

Core ファイルの出力設定は以下の通りです。

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 %s：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
Core ファイル出力サイズ(Soft)	0	既定値では Core ファイルは出力されません
Core ファイル出力サイズ(Hard)	unlimited	出力サイズ上限は無制限

3.2 HPC 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「Amazon DCV(Desktop Cloud Visiualization)」の利用準備について説明します。以下の URL から「Amazon 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.amazondcv.com/latest.html

3.2.2 Amazon 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、「Amazon DCV サーバ」の仮想セッションを作成します。

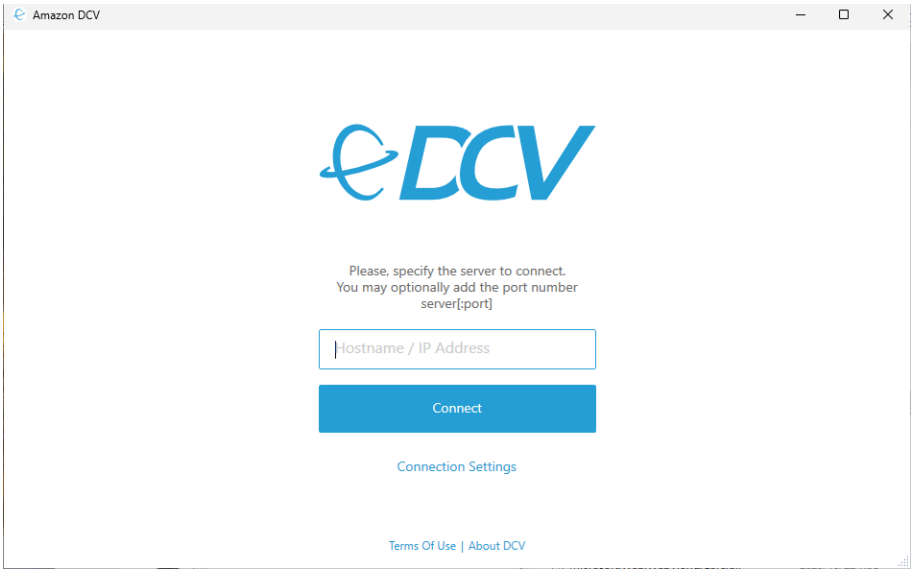
```
$ dcv create-session --type=virtual セッション名
(仮想セッションを作成します)
$ dcv list-sessions
(作成した仮想セッションを確認します)
```

HPC フロントエンドは 2 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. Amazon 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。

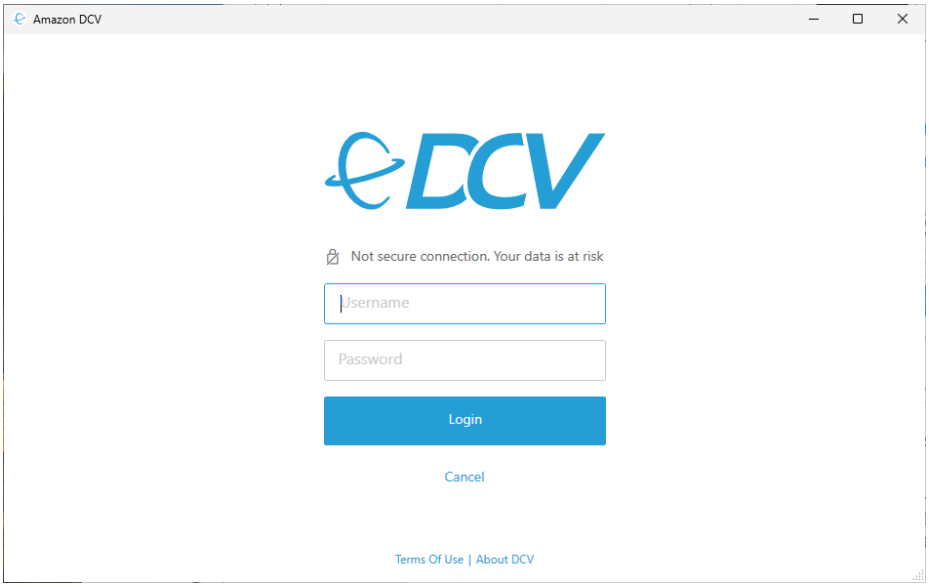
3.2.3 Amazon DCV クライアントの起動

「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。

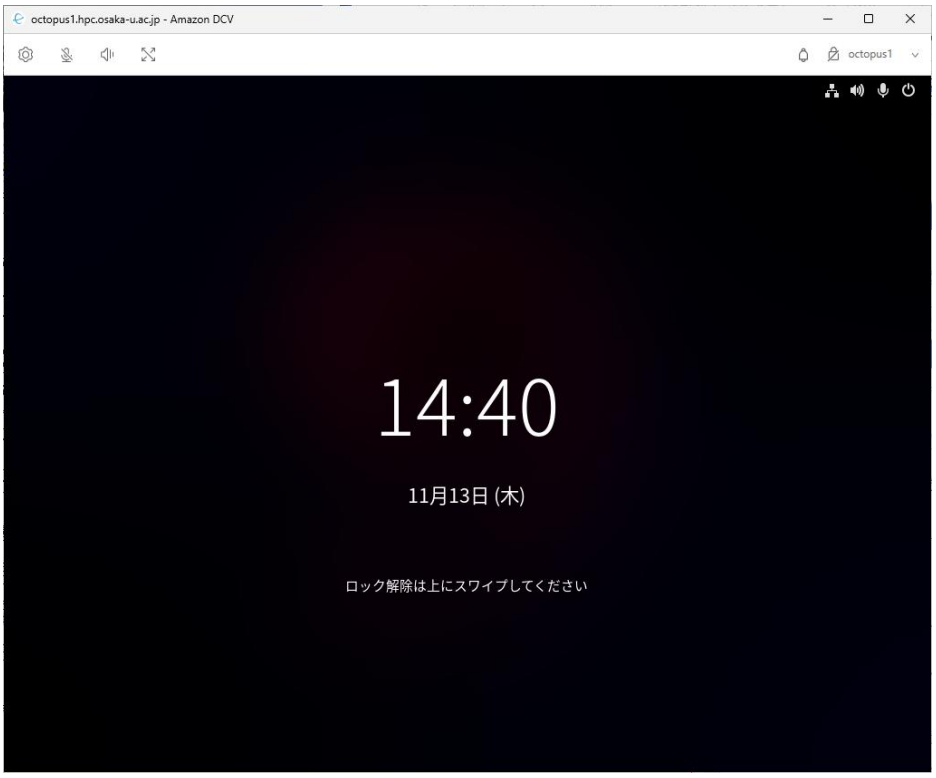
「3.2.2. Amazon DCV サーバの仮想セッション作成」でセッションを作成したサーバに接続します。例えば、仮想セッションを作成したサーバが「octopus1」、セッション名が「mysession」であった場合、「octopus1.hpc.osaka-u.ac.jp:5901/#mysession」と入力し、[Connect]をクリックします。



HPC フロントエンド上の「Amazon 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。



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 Amazon DCV サーバの仮想セッション削除

Amazon 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、Amazon 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。

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占有型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 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 oneAPI	Intel MPI
(なし)	BaseGCC	GNU Compiler	Open MPI

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/2025   BaseCPU/2025   BaseExtra/2025   BaseGCC/2025   BaseJDK/2025
BaseJulia/2025   BasePy/2025   BaseR/2025

$ 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。
- GNU コンパイラ
  「4.3 GNU Compiler Collection によるコンパイル」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
Python3	BasePy	3.13.5	Python 言語向け	
R	BaseR	4.5.1	R 言語向け	
JAVA	BaseJDK	21.0.8	JAVA 言語向け	OpenJDK
Julia	BaseJulia	1.11.6	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
```

Python 言語に関して、「4.5 Python の利用方法」に詳細を記載します。

4.1.2 ライブラリ・言語モジュールの利用

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	GCC	Py	R
Intel MKL (※)	○	-	-	-
GNU Scientific Library	-	○	-	-
HDF5	○	-	-	-
NetCDF	○	-	-	-
Parallel netcdf	○	-	-	-
Keras	-	-	○	-
PyTorch	-	-	○	-
TensorFlow	-	-	○	-
pbdR	-	-	-	○

※ BLAS, LAPACK, ScaLAPACK は「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、「4.2 汎用 CPU 計算環境向けプログラムのコンパイル」の「ライブラリの利用」を参照ください。

4.2 汎用 CPU 計算環境向けプログラムのコンパイル

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
```

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 oneAPI
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。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

$ ifx -o sample sample.f90
$ icx -o sample sample.c
$ icpx -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 oneAPI
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 BaseCPU/2025

$ ifx -o sample_omp -qopenmp sample_omp.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	mpiifx
C	mpiicx
C++	mpiicpx

以下はコンパイル例です。

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

```
$ mpiifx -o sample_mpi sample_mpi.f90
$ mpiicx   -o sample_mpi sample_mpi.c
$ mpiicpx -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
$ 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 BaseCPU/2025
$ icx -qmkl=parallel sample.c                    # 動的リンク
$ icx -qmkl=parallel -static-intel sample.c      # 静的リンク

# MPI 並列実行で MKL をリンク
$ module load BaseCPU/2025
$ 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。

- Intel Math Kernel Library - Documentation
  https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-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。HDF5 は「並列化対応」と「C+ライブラリ対応」の 2 つが利用可

能です。それぞれのライブラリのリンク例です。

・並列化対応

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

・C++ライブラリ対応

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

・C＋＋言語

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

・Fortran 言語

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

・C＋＋言語

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

・Fortran 言語

```
$ module load BaseCPU
$ module load pnetcdf-fortran
$ mpiifx -o pncdf-sample pncdf-sample.f90 -lpnetcdf
```

4.3 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
```

4.3.1 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	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
```

4.3.2 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	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
```

4.4 コンテナの利用方法

本システムでは、Apptainer(Singularity)を用いた、コンテナによるプログラム実行環境を整備することが可能です。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。

4.4.1 コンテナイメージの準備

コンテナイメージは、OCTOPUS 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 OCTOPUS 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。

- **OCTOPUS ローカルレジストリからイメージを取得する**

  OCTOPUS 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。

  test コンテナイメージは、実運用に使用することは想定していないビルドテスト用のイメージになります。

```
$ apptainer build test.sif ¥
```

```
        oras://cntm:5000/master_image/test:1
```

コマンドの書式は以下の通りです。

```
$ apptainer 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。

- **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、apptainer 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、RockyLinux 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。

```
$ apptainer build rocky9.sif docker://rockylinux:9
```

コマンドの書式は以下の通りです。

```
$ apptainer build <イメージファイル名> docker://<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 rocky9.sif）が作成されます。

4.4.2 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。
OCTOPUS 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
$ apptainer build -f --sandbox --fix-perms test test.sif
```

・拡張領域で sandbox を作成する場合

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

apptainer コマンドの書式は以下の通りです。

```
$ apptainer 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。

```
$ apptainer run -f -w test
```

コマンドの書式は以下の通りです。

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

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

```
Apptainer>
```

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。

```
Apptainer> exit
```

(3) イメージのファイル化

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。

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

コマンドの書式は以下の通りです。

```
$ apptainer 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。

コマンドの書式は以下の通りです。

```
$ apptainer 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

最初に、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
3
4  %files
5          ./test.conf /opt/test.conf
6          ./test_start.sh /opt/test_start.sh
7
8
9  %post
10         dnf install -y net-tools
11         chmod 755 /opt/test_start.sh
12
13 %runscript
           /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） %files

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、Apptainer のマニュアルをご参照ください。

https://apptainer.org/docs/user/latest/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。最初に、ビルドに使用する def ファイルを作成します。

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

コマンドの書式は以下の通りです。

```
$ apptainer 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。

4.5 Python の利用方法

本システムでは、Python 言語環境として、Python3 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
```

4.5.1 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。

```
# モジュールの読み込み
$ module load BasePy

# 対話モードの起動
$ python3
Python 3.13.5 (main, Aug 13 2025, 18:27:42) [Clang 21.0.0git (icx 2025.2.0.20250605)] on linux
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()
$
```

4.5.2 プログラム(スクリプト)実行

```
# モジュールの読み込み
$ module load BasePy

# プログラム実行
$ python3 sample.py
```

4.5.3 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、pip3 コマンドを利用します。pip3 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
pip3 list	インストール済み Python モジュール一覧の表示
pip3 install [pymod]	Python モジュールのインストール
pip3 show [pymod]	Python モジュールの詳細表示
pip3 uninstall [pymod]	Python モジュールのアンインストール

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。

```
# 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
```

4.5.4 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とは

OCTOPUS 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、OCTOPUS 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。
OCTOPUS 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を以下に記載します。

```
$ qlogin -q INT --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 **1234.oct** submitted to queue: INT.
Waiting for **1234.oct** 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 OCT
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=256
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@xxx.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 **1234.oct** submitted to queue: OCT.

qsub コマンドの主なオプションは以下の通りです。

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
-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。 OpenMPI を利用する場合は、-T openmpi を指定してください。

ノード数の上限を超えたジョブを投入した場合は、
実行キュー(DBG,VA,DPS)と転送キュー(OCT,OCT-H,OCT-S)で挙動が異なります。

【実行キューの場合】
ジョブ投入時にエラーメッセージが表示されます。

【転送キューの場合】
標準エラー出力ファイルにエラーメッセージが出力されます。

以下、標準エラー出力ファイルの出力例になります。
XXXXX は投入したリクエストのリクエスト ID になります。

```
[<ユーザ名>@octopus1 <ディレクトリ>]$ qsub run-TESTQ.sh Request XXXXX.oct
submitted to queue: OCT.
[<ユーザ名>@octopus1 <ディレクトリ>]$ cat run-TESTQ.sh.eXXXXX Request could
not be queued to execution queues Request deleted.

The request could not be queued to any execution queues because of the
following reason(s):

Exceeded resource limit.
```

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。。
各コマンドで表示されるバッチリクエストの状態は下記のとおりです。
SUSPENDING、SUSPENDED、RESUMING の各状態はいずれも“SUS”と表示されます。

表示	状態	状態概要
QUE	QUEUE	実行待ち状態 スケジュール対象となります
RUN	RUNNING	リクエスト内のジョブ実行中です。
POR	POST-RUNNING	リクエスト内ジョブの後処理中です。
PRR	PRE-RUNNING	リクエスト内ジョブの前処理中です。
EXT	EXITING	実行結果ファイルの転送中です。
STG	STAGING	バッチジョブの生成および クライアントから実行ホストへファイル転送中です。
HLD	HELD	リクエストはスケジューリング対象から外されています。 スケジューラからのラン要求またはリスタート要求を受け付けない状態です。
WAT	WAITING	実行時刻待ち合わせ中です。
SUS	SUPENDING	対象リクエスト内の全ジョブ停止待ち合わせ中です。
SUS	SUSPENDIED	対象リクエスト内の全ジョブが停止中です。
SUS	RESUMINNG	対象リクエスト内の全ジョブの再開待ち合わせ中です。
TRS	TRANSITING	他キューへリクエスト転送中です。
ARI	ARRIVING	転送キューからのリクエスト受信中です。
HOL	HOLD	終了型チェックポイントの採取中です。 リスタートファイル採取後、ジョブは終了します。

MIG	MIGRATING	対象リクエスト内に、あるジョブサーバの管理下から 別のジョブサーバへ移動中のジョブがある状態です。

(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 実行時のオプションの有無で、出力されるヘッダー情報に差異があります。

```
$ sstat
```

以下は表示例です。リクエスト ID とユーザ名は仮の値に置き換えています。

リクエスト名、ユーザ名はオプション付与した場合、無しの場合より長く表示されます。

開始時刻は、オプション無しの場合は実行中でも時刻が表示されますが、オプション有の場合は実行中であればメッセージが変化します。

オプション無

```
RequestID          ReqName    UserName Queue        Pri STT Date(PLANNED START)
---------------- -------- -------- -------- ---- --- -------------------
xxxxx.oct          test.s yyyyy O1              0 RUN 2026-01-14 20:34:13
```

オプション有

```
RequestID                                ReqName                UserName          Queue
Pri STT PlannedStartTime
------------------------------------ -------------------- --------------- --------------- --------------
--- --- -------------------
xxxxx.oct                                test.sh                yyyyy             O1
```

```
0.5002/   0.5002 RUN Already Running...
```

自分および、自分が所属しているグループのユーザが投入したリクエストのスケジューリング状態を表示する場合

HPCI 利用者の場合は、自分が所属している課題のユーザが投入したバッチリクエストの状態を表示します。

```
$ sstatgroup
```

現在投入されている全バッチリクエストのスケジューリング状態を表示する場合

自分と、自分が所属しているグループのユーザが投入したバッチリクエスト（HPCI 利用者の場合は自分が所属している課題）以外は、ユーザ名とスクリプト名をマスクした状態で表示します。

以下は出力例です。リクエスト ID は仮の値に置き換えています。

```
$ ssstatall
```

```
RequestID         ReqName   UserName     Queue                        Pri STT PlannedStartTime
--------------- -------- ---------- ---------- ---------------- --- -------------------
xxxxx.oct         -------- ---------- O1            0.5002/ 0.5002      RUN Already Running...
```

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) バッチジョブの解除

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 %s：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

OCTOPUS 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。

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	OCT	120 時間	32,768core (256c/ノード)	95TiB (760GB/ノード)	128	ノード内は占有利用
	OCT-H	120 時間	32,768core (256c/ノード)	95TiB (760GB/ノード)	128	ノード内は占有利用 ※1
	OCT-S	120 時間	128core	380GiB	1	※2
	DBG	10 分	512core (256c/ノード)	1,520GB (760GB/ノード)	2	デバッグ用
	INT	10 分	512core (256c/ノード)	1,520GB (760GB/ノード)	2	インタラクティブ利用
	VA	120 時間	256core	760GB	1	疑似量子ア

						ニーリング用
占 有 利用	myOCT	無制限	256core × 占有ノード数	760GB × 占有ノード数	占有数	別途申請

※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　　：OCT
- 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 OCT
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。

- 11 行目
  ジョブスクリプト実行ディレクトリに移動します。
  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。

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　　：OCT

- 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：256（128 コア×2CPU×1 ノード）

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q OCT
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -v OMP_NUM_THREADS=256
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。

- 13 行目
  ジョブスクリプト実行ディレクトリに移動します。
  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。

5.3.3 MPI 利用方法

4 ノード内で 1,024 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　　：OCT
- 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　：1,024（128 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 OCT
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 1024 ./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。

- 14 行目
  ジョブスクリプト実行ディレクトリに移動します。
  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。

- 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。

5.3.4 MPI＋ノード内並列 利用方法

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 256 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　　：OCT
- 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 ：1,024（128 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
- スレッド数　　：256（環境変数 OMP_NUM_THREADS で指定）

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q OCT
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=256
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。

- 15 行目
  ジョブスクリプト実行ディレクトリに移動します。
  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。

- 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。

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。

OCTOPUS 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　　：OCT
- 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 ：128（64 コア×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 OCT
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。

  14 行目
  ジョブスクリプト実行ディレクトリに移動します。
  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。

- 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　　：OCT
- 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　：2（2 コア×1 ノード）
- MPI ライブラリ：IntelMPI

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

- 12 行目
  ジョブスクリプト実行ディレクトリに移動します。
  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。

- 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 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。

5.4.1 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、Rocky Linux のイメージファイル(rocky9.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、rocky9.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。

```
$ apptainer exec rocky9.sif hostname
```

コマンドの書式は以下の通りです。

```
$ apptainer 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

```
$ apptainer exec --env MYVAR="My Value!" rocky9.sif myprog.exe
```

・ --env-file オプションによる一括指定

```
$ cat myenvfile
```

```
MYVAR="My Value!"
$ apptainer exec --env-file myenvfile rocky9.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://apptainer.org/docs/user/latest/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：/octfs/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。

```
$ apptainer exec rocky9.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 /octfs/work/(グループ名)/(利用者番号)
$ apptainer exec --bind `pwd` rocky9.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://apptainer.org/docs/user/latest/bind_paths_and_mounts.html

5.4.2 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　　：OCT
- 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：256（128 コア×2CPU×1 ノード）

```
1   #!/bin/bash
2   #------- qsub option -----------
3   #PBS -q OCT
4   #PBS --group=G01234
5   #PBS -l elapstim_req=00:30:00
6   #PBS -v OMP_NUM_THREADS=256
7
8   #------- Program execution -----------
9
10  cd $PBS_O_WORKDIR
11  apptainer 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、apptainer 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。

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導入しています。

6.1 アプリケーション一覧

以下、利用可能ホストの説明です。
octopusX　　＝HPC フロントエンド
voctropusXX＝占有型フロントエンド

- 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		
	octopusX	voctopusXX	汎用 CPU
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		
	octopusX	voctopusXX	汎用 CPU
ABINIT-MP	○	○	○
ADIOS	○	○	○
CTFFIND	○	○	○
FLASHcode	○	○	○
FreeFem++	○	○	○
GAMESS	○	○	○
GENESIS	○	○	○
GROMACS	○	○	○
Gnuplot	○	○	-
ImageMagick	○	○	-
LAMMPS	○	○	○
MotionCor3	○	-	-
NCView	○	○	○
OpenFOAM	○	○	○
ParaView	○	○	-
Quantum ESPRESSO	○	○	○
Relion	○	○	-

ResMAP	○	○	-
VisIt	○	○	○
SMASH	○	○	○
NEC Vector Annealing	-	-	○

6.2 ISV アプリケーション利用方法

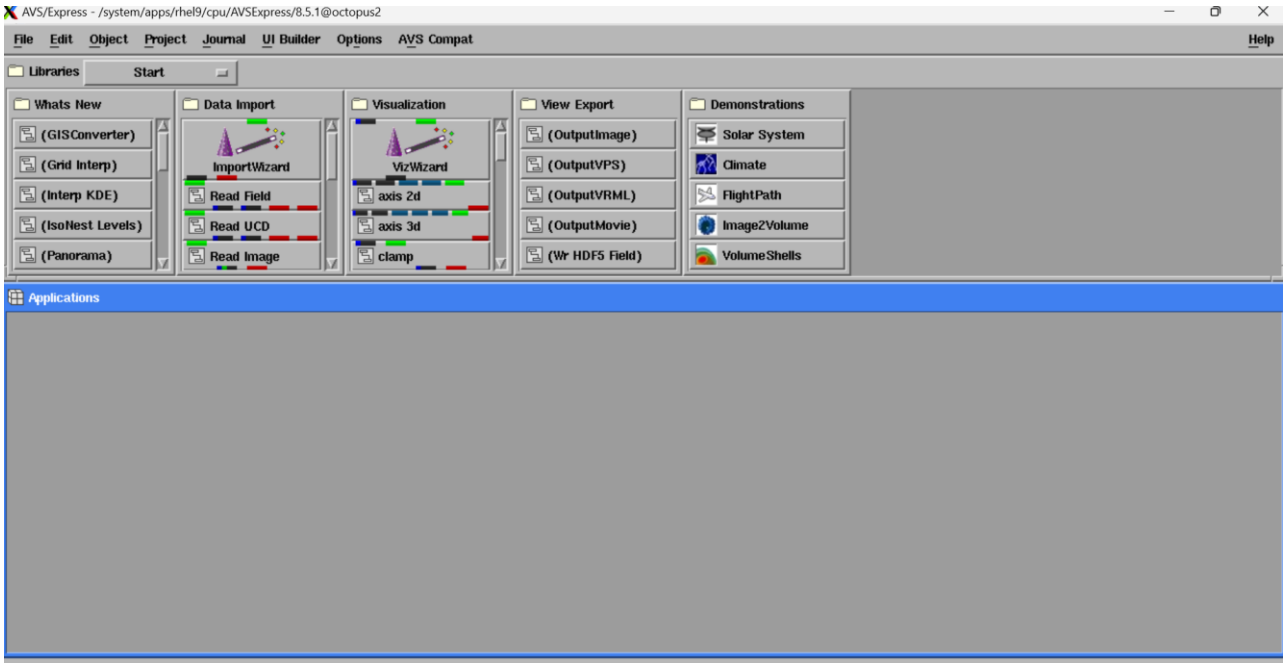
OCTOPUS システムで使用可能な ISV アプリケーションの利用方法について説明します。

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。OCTOPUS 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。



AVS/Express をお手元の Windows 端末にインストールしてご利用いただく場合、AVS 利用申請書を提出いただいて貸出メディアをお渡しします。 貸出メディアに添付されているセットアップ方法に従ってインストールし、デスクトップのアイコンをダブルクリックして起動してください。
ライセンスサーバは SQUID システムで提供しています。ライセンスサーバとポート番号は以下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 を提供しています。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 OCT
#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
g16 <  入力ファイル  >& result
```

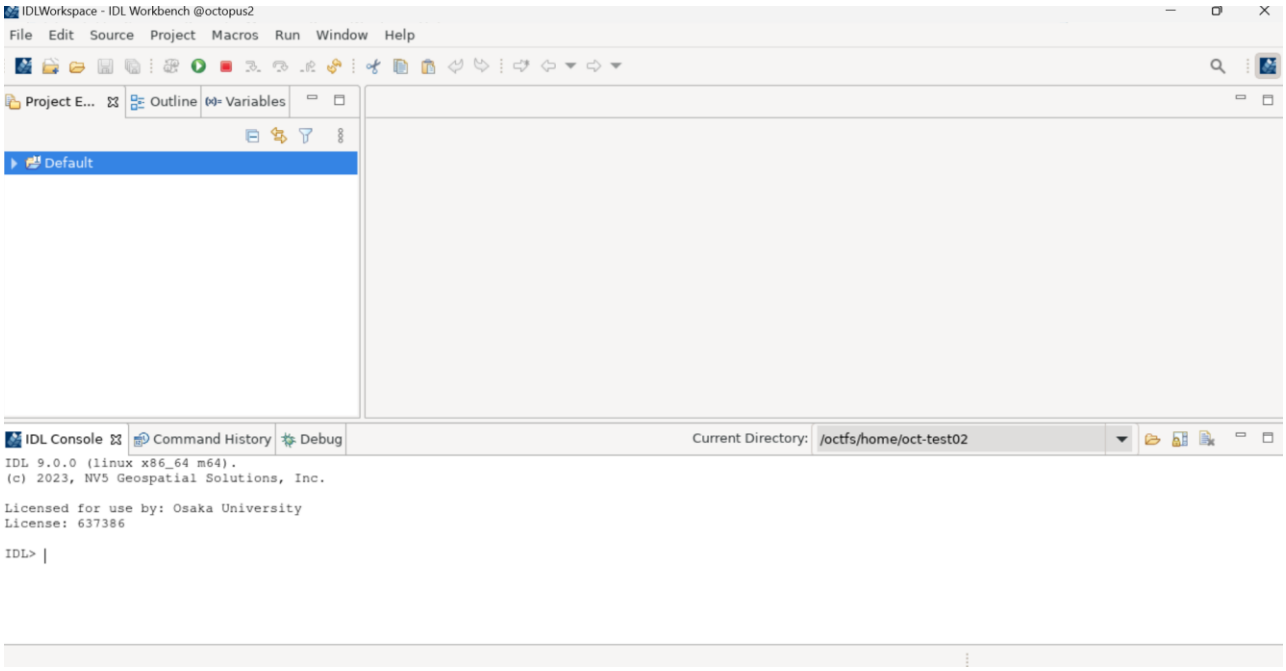
6.2.3 IDL

「IDL(Interractive Data Language)」はデータ解析、可視化、クロスプラットフォーム対応アプリケーション開発に適したソフトウェアでバージョン 9.0 を提供しています。

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

OCTOPUS システムで使用可能な主な OSS アプリケーションの利用方法について説明します。

6.3.1 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。

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

<CPU ノード>

```
#!/bin/bash
#PBS -q OCT-S
#PBS -l cpunum_job=20
#PBS --group= <グループ名>
```

```
#PBS -l elapstim_req=01:00:00
#PBS -b 1
#PBS -T intmpi

module load BaseCPU
module load BaseApp
module load abinit-mp/2.8

cd ${PBS_O_WORKDIR}
export MPI_PROC=20

#------- ABINIT-MP option -------#
BINARY_NAME=abinitmp

#------- Input file -------#
cp /system/apps/rhel9/cpu/ABINIT-MP/InteloneAPI2025.2.0/2.8/examples/Gly5/gly5-mpi.ajf .
FILE_NAME=gly5-mpi
AJF_NAME=${FILE_NAME}.ajf

#------- Output file -------#
NUM_CORE=_001n-020p-ver2rev8
OUT_NAME=${FILE_NAME}${NUM_CORE}.log

#------- Program execution -------#
mpiexec -n ${MPI_PROC} ${BINARY_NAME} < ${AJF_NAME} > ${OUT_NAME}
```

6.3.2 ADIOS

「ADIOS」は大規模科学計算における高速かつ柔軟な並列 I/O を提供するフレームワークです。
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 OCT
#PBS --group= <グループ名>
```

```
#PBS -T intmpi
#PBS -l elapstim_req=24:00:00
#PBS -b 1

module load BaseApp
module load adios/1.13.1

cd ${PBS_O_WORKDIR}

cp                                                                    -r
/system/apps/rhel9/cpu/ADIOS/InteloneAPI2025.2.0/1.13.1/examples/Fortran/scalars .
cd scalars

mpiifx scalars_write.F90 -I$CPATH -ladiosf -o scalars_write
mpiifx scalars_read.F90   -I$CPATH -ladiosf -o scalars_read

mpirun -np 2 ./scalars_write
mpirun -np 2 ./scalars_read
```

利用方法は、以下マニュアルを参照ください。
https://users.nccs.gov/~pnorbert/ADIOS-UsersManual-1.13.1.pdf

6.3.3 CTFFIND

「CTFFIND」は電子顕微鏡画像写真からのパラメータを高精度に推定するソフトウェアです。
サンプルデータで実行する場合、事前にデータをコピーしてデータパスの修正を行います。

```
$ cp -r /system/apps/rhel9/cpu/CTFFIND/InteloneAPI2025.2.0/4.1.14/examples .
$ cd examples
$ vi ctffind_input.txt
  →1 行目の「/USER_DIR/cell-025.mrc」にある「USER_DIR」を「cell-025.mrc」を
    格納しているフルパスに修正してジョブを投入してください。
$ qsub <ジョブスクリプト>
```

以下はサンプルデータで実行する例です。

```
#!/bin/bash
#PBS -q OCT
#PBS --group= <グループ名>
```

```
#PBS -l elapstim_req=01:00:00

module load BaseApp
module load CTFFIND/4.1.14

cd $PBS_O_WORKDIR

ctffind < ctffind_input.txt
```

6.3.4 FLASHcode

「FLASHcode」は適応メッシュ細分化を用いた並列マルチフィジックスシミュレーションコードです。
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 OCT-S
#PBS -l cpunum_job=4
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 1
#PBS -T intmpi

module load BaseApp
module load FLASHcode/4.8

cd $PBS_O_WORKDIR
cp -p /system/apps/rhel9/cpu/FLASHcode/InteloneAPI2025.2.0/4.8/examples/flash.par .

mpirun -np 4 flash4
```

6.3.5 FreeFem＋＋

「FreeFem++」は有限要素法（FEM）を用いて偏微分方程式を解くためのオープンソースのソフトウェアです。
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 OCT
#PBS --group= <グループ名>
```

```
#PBS -l elapstim_req=24:00:00
#PBS -b 1
#PBS -T intmpi

module load BaseApp
module load FreeFemXX/4.15

cd ${PBS_O_WORKDIR}

cp                                                                    -r
        /system/apps/rhel9/cpu/FreeFemXX/InteloneAPI2025.2.0/4.15/share/FreeFEM/4.1
        5/examples .
cd examples
./go_CheckAllEdp.sh
```

6.3.6 GAMESS

「GAMESS」は分子の電子構造計算を行う高機能な量子化学プログラムです。
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 OCT
#PBS --group= <グループ名>
#PBS -l elapstim_req=24:00:00
#PBS -b 1
#PBS -T intmpi

cd ${PBS_O_WORKDIR}

module load BaseApp
module load GAMESS/v2024.2.1

export SCR=${PBS_O_WORKDIR}
export USERSCR=${PBS_O_WORKDIR}
```

```
cp
        /system/apps/rhel9/cpu/GAMESS/InteloneAPI2025.2.0/v2024.2.1/tests/standard/e
        xam01.inp .

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>
```

6.3.7 GENESIS

「GENESIS（GENeralized-Ensemble SImulation System）」は、生体分子システムの分子動力学シミュレーションを実行するためのアプリケーションです。
計算精度により読み込む Module ファイルが異なります。

倍精度	混合精度	単精度
2.1.5.double	2.1.5.mixed	2.1.5.single

（例）倍精度の場合

```
#!/bin/bash

#PBS -q OCT
#PBS --group= <グループ名>
#PBS -l cpunum_job=8
#PBS -l elapstim_req=24:00:00
#PBS -b 1
#PBS -T intmpi

cd ${PBS_O_WORKDIR}

module load BaseApp
module load GENESIS/2.1.5.double

cp                                                                      -r
        /system/apps/rhel9/cpu/GENESIS/InteloneAPI2025.2.0/2.1.5/double/tests/regressi
        on_test .
```

```
cd regression_test


binary=`which spdyn`
./test_rpath.py "mpirun -n 8 ${binary}"
```

6.3.8 Gnuplot

「Gnuplot」は科学技術計算結果を 2D・3D で可視化するためのプロットツールです。
フロントエンドにログイン後、以下を実行します。

```
$ gnuplot
```

6.3.9 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
```

6.3.10 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、ATC、AWPMD、BOCS、BODY、CG-DNA、CG-SPICA、CLASS2、COLLOID、
COLVARS、COMPRESS、CORESHELL、DIFFRACTION、DIPOLE、DPD-MESO、DRUDE、
EFF、ELECTRODE、EXTRA-COMPUTE、EXTRA-DUMP、EXTRA-FIX、EXTRA-MOLECULE、
EXTRA-PAIR、FEP、GRANULAR、INTEL、INTERLAYER、KSPACE、LATBOLTZ、MANIFOLD、
MANYBODY、MC、MDI、MGPT、MISC、MOFFF、MOLECULE、NETCDF、OPENMP、OPT、
ORIENT、PERI、PHONON、POEMS、PYTHON、QEQ、QTB、REAXFF、REPLICA、RIGID、
SHOCK、SMTBQ、SPH、SPIN、SRD、TALLY、UEF、VORONOI
```

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 OCT
#PBS -l cpunum_job=256
#PBS --group= <グループ名>
#PBS -l elapstim_req=01:00:00
#PBS -b 1
#PBS -v OMP_NUM_THREADS=128
```

```
module load BaseApp
module load lammps/22Jul2025
cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 2 lmp < ./in.lj
```

6.3.11 MotionCor3

「MotionCor3」はマルチ GPU を用いてクライオ電子顕微鏡・トモグラフィー画像のビーム誘起運動をピクセル単位で異方性補正するソフトウェアです。
フロントエンドで実行してください。

```
module load BaseApp
module load MotionCor3/1.0.1

MotionCor3 -InMrc test_boxmovie.mrcs -OutMrc test_boxmovie_corrected.mrc
```

6.3.12 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＞

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

cd ${PBS_O_WORKDIR}

module add BaseApp
module load OpenFOAM/v2412

cp -r /system/apps/rhel9/cpu/OpenFOAM/InteloneAPI2025.2.0/v2412/OpenFOAM-v2412/tutorials/incompressible/icoFoam/cavity/cavity .
cd cavity

blockMesh
icoFoam
```

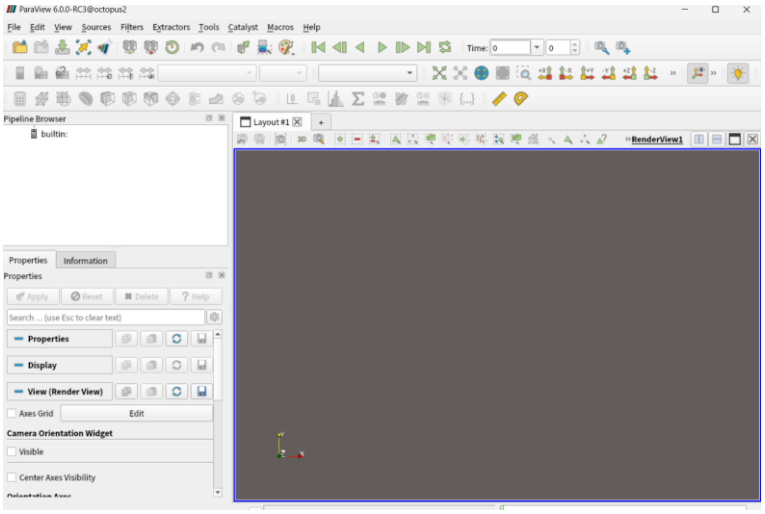
6.3.13 ParaView

「ParaView」は大規模データの並列可視化を可能にするオープンソースソフトウェアです。

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/6.0.0
$ paraview
```

以下のように ParaView の操作画面が表示されます。

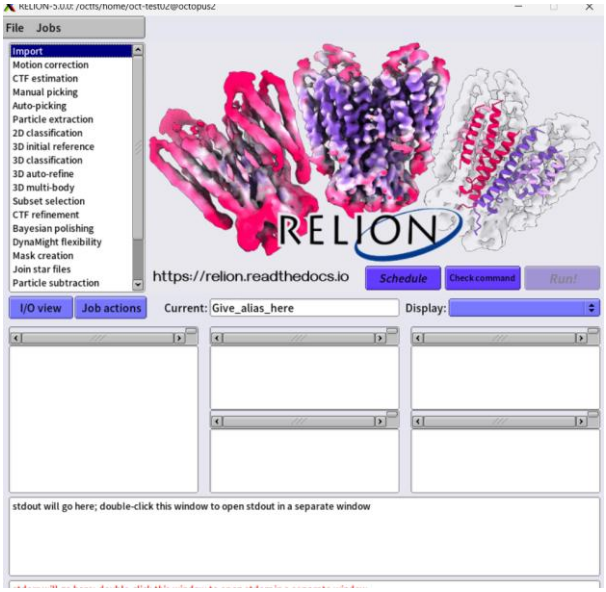


6.3.14 Relion

「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/5.0.0
$ relion
```

以下のように Relion の操作画面が表示されます。



6.3.15 ResMAP

「ResMap」は低温電子顕微鏡（Cryo-EM）などを用いた単粒子解析において、構造の分解能マップ（Resolution Map）を計算・可視化するためのソフトウェアです。

フロントエンドで「ResMAP」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 ResMap/1.1.4
$ ResMap
```

※起動時にフォント関連のエラー、ワーニングが出力されますが実行には問題ありません。

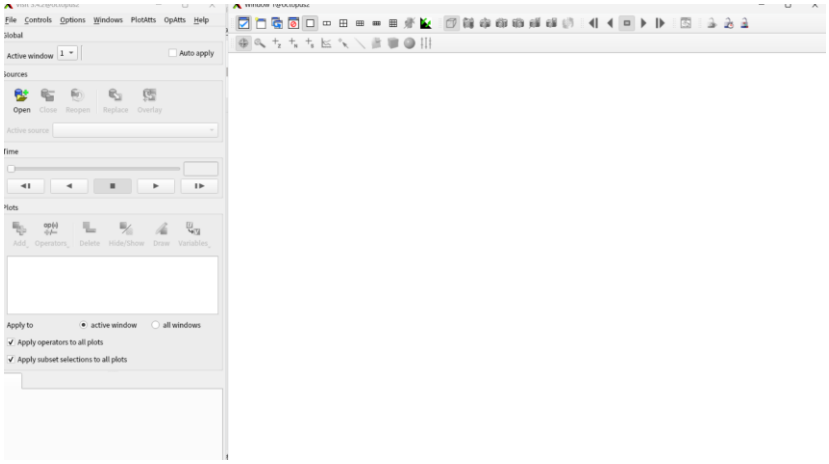
実行に問題がありましたらご連絡をお願いします。

6.3.16 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/3.4.2
$ visit
```

以下のように VisIt の操作画面が表示されます。



6.3.17 SMASH

「SMASH」は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 OCT
#PBS --group=<グループ名>
#PBS -l cpunum_job=4
#PBS -l elapstim_req=24:00:00
#PBS -b 1
#PBS -T intmpi

cd ${PBS_O_WORKDIR}

module load BaseApp
module load smash/3.0.2

cp /system/apps/rhel9/cpu/SMASH/InteloneAPI2025.2.0/3.0.2/example/b3lyp-energy.inp .
```

```
mpirun ${NQSV_MPIOPTS} -np 4 smash < b3lyp-energy.inp
```

6.3.18 NEC Vector Annealing

「NEC Vector Annealing」は x86 上にシミュレーテッドアニーリングで求解するベクトルアニーリング（VA）エンジンを実装し、組合せ最適化問題を定式化して解くソフトウェアです。

NEC Vector Annealing 実行に際して必要となる「numpy」、「pyqubo」は「BasePy」をロードすることにより利用可能となります。
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 VA
#PBS --group=<グループ名>
#PBS -l elapstim_req=00:05:00
#PBS -b 1

#PBS -v OMP_NUM_THREADS=8

cd ${PBS_O_WORKDIR}

module load BasePy
module load annealing/4.0.0

python /opt/va/V4.0.0X/samples/TSP/python/TspLib.py eil51.tsp
```

※ 「eli51.tsp」は Heidelberg University の下記サイト

```
http://comopt.ifi.uni heidelberg.de/software/TSPLIB95/tsp/
```

から「ALL_tsp.tar.gz」をダウンロードして解凍し、ジョブスクリプトと同じディレクトリに格納してください。「NEC Vector Annealing」では“EDGE_WEIGHT_TYPE : EUC_2D”問題のみ対応しています。また 100 都市以上のような大規模な問題ファイルは QUBO が大きくなり、Pyqub で生成できない可能性がありますので、使用しないようお願いいたします。

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

OCTOPUS 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、OCTOPUS 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。

/octfs/home/(利用者番号)/OnionWeb/

例：利用者番号"user001"の場合　/octfs/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。

https://oct-dtg1.hpc.osaka-u.ac.jp

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。

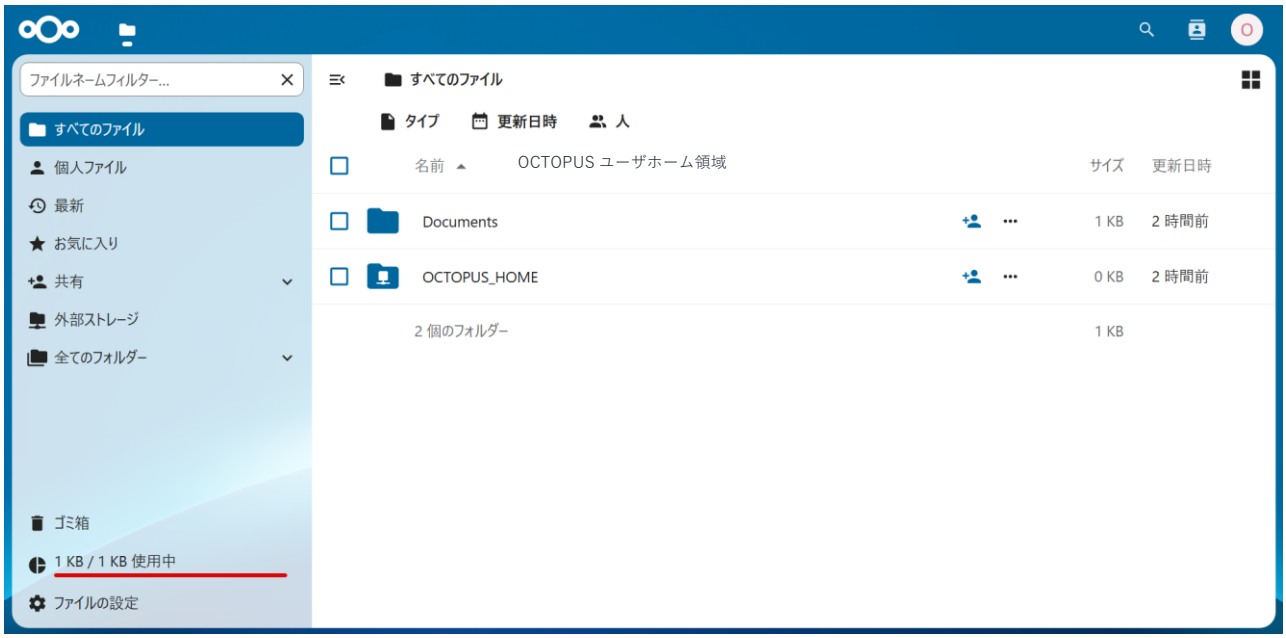
二要素認証の設定を行います。「TOTP(Authenticator app)」を選択してください。

Nectcloud 用 QR コードが表示されますので、ご自身の端末、スマートフォンにインストールした二要素認証アプリケーションで読み込んでください。

「TOTP（Authenticator app)」を選択してください。

二要素認証アプリケーションに表示された 6 桁の認証コードを入力してください。

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「OCTOPUS_HOME」として表示されます。

7.1.2 基本的な使い方

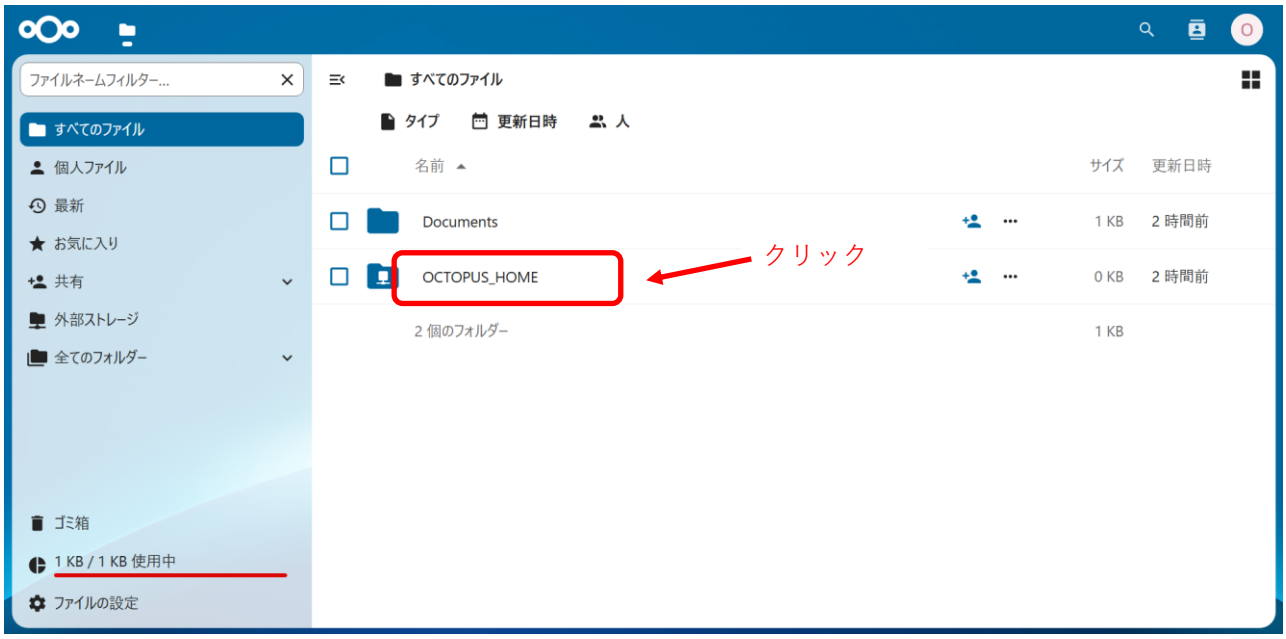
➢ **アップロード方法**

ファイルのアップロード先は、「OCTOPUS_HOME」、または後述する「7.1.4 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「OCTOPUS_HOME」の直下にファイルをアップロードします。

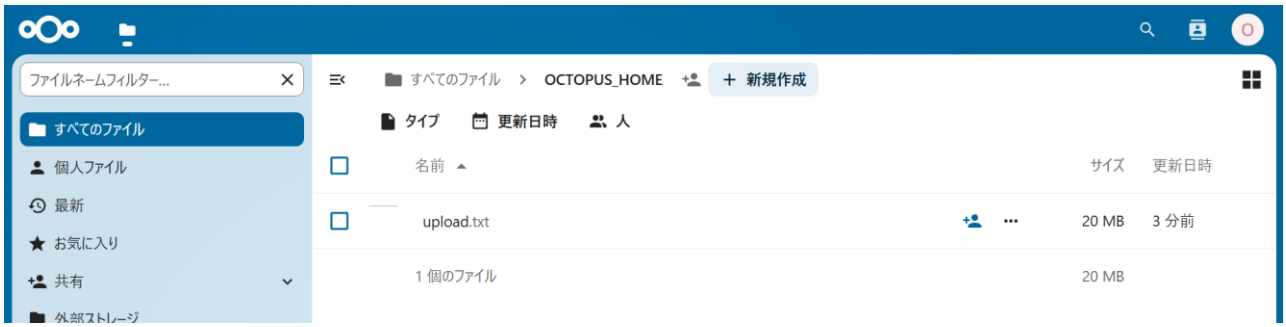
「OCTOPUS_HOME」をクリックします。



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。

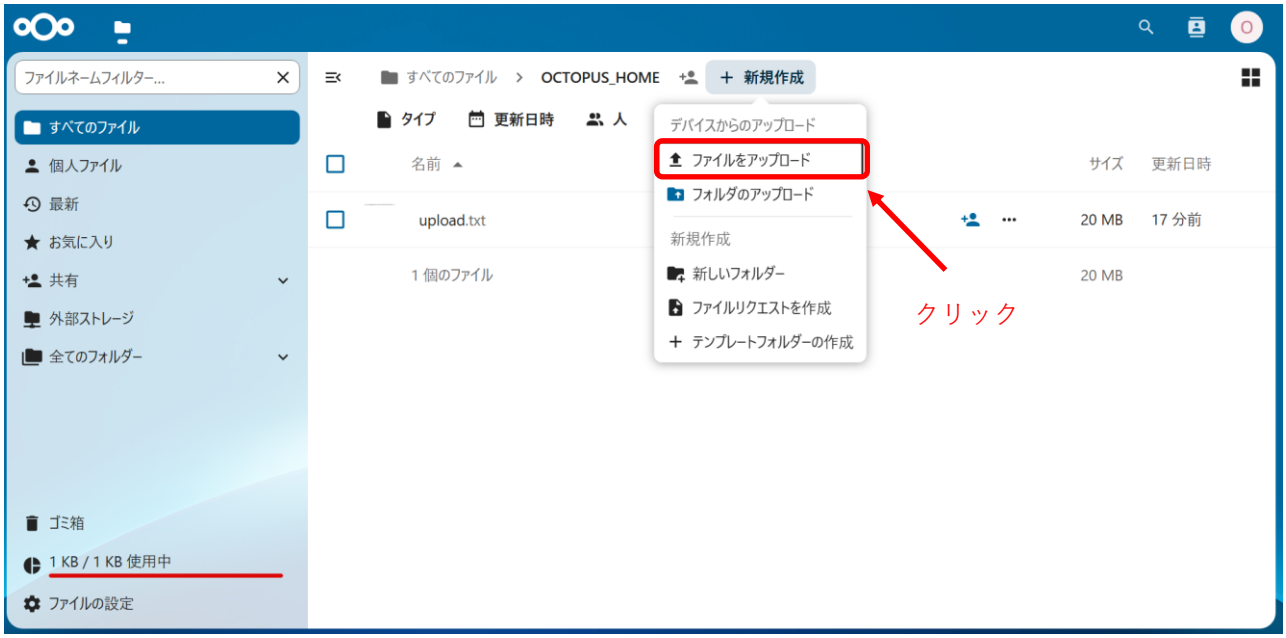


(2) 参照によるアップロード

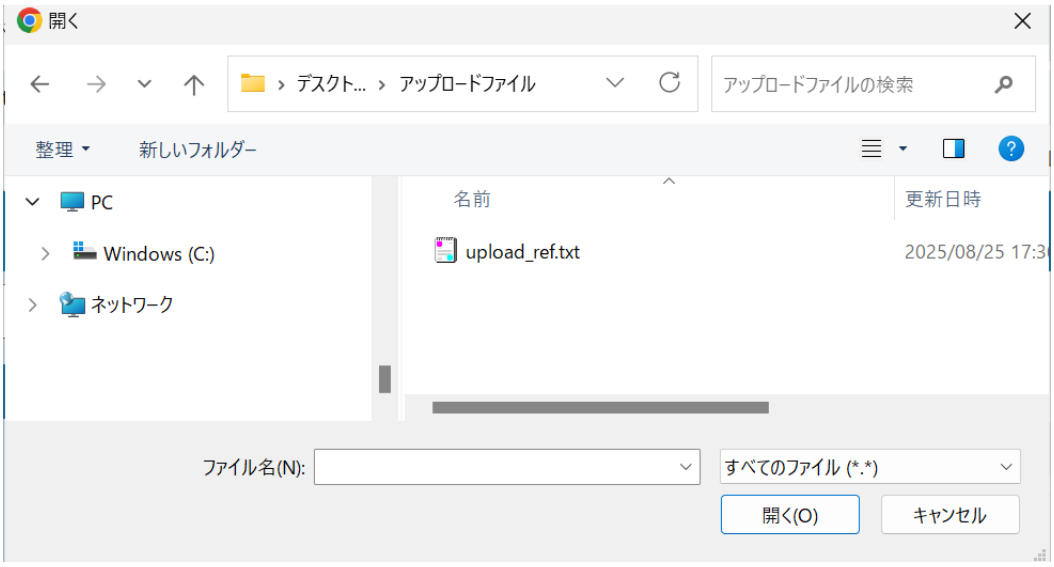
アップロードしたい場所を開き、アップロードしたいファイルを選択します。
以下の例では「OCTOPUS_HOME」の直下にファイルをアップロードします。

「OCTOPUS_HOME」をクリックします。

「+新規作成」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。

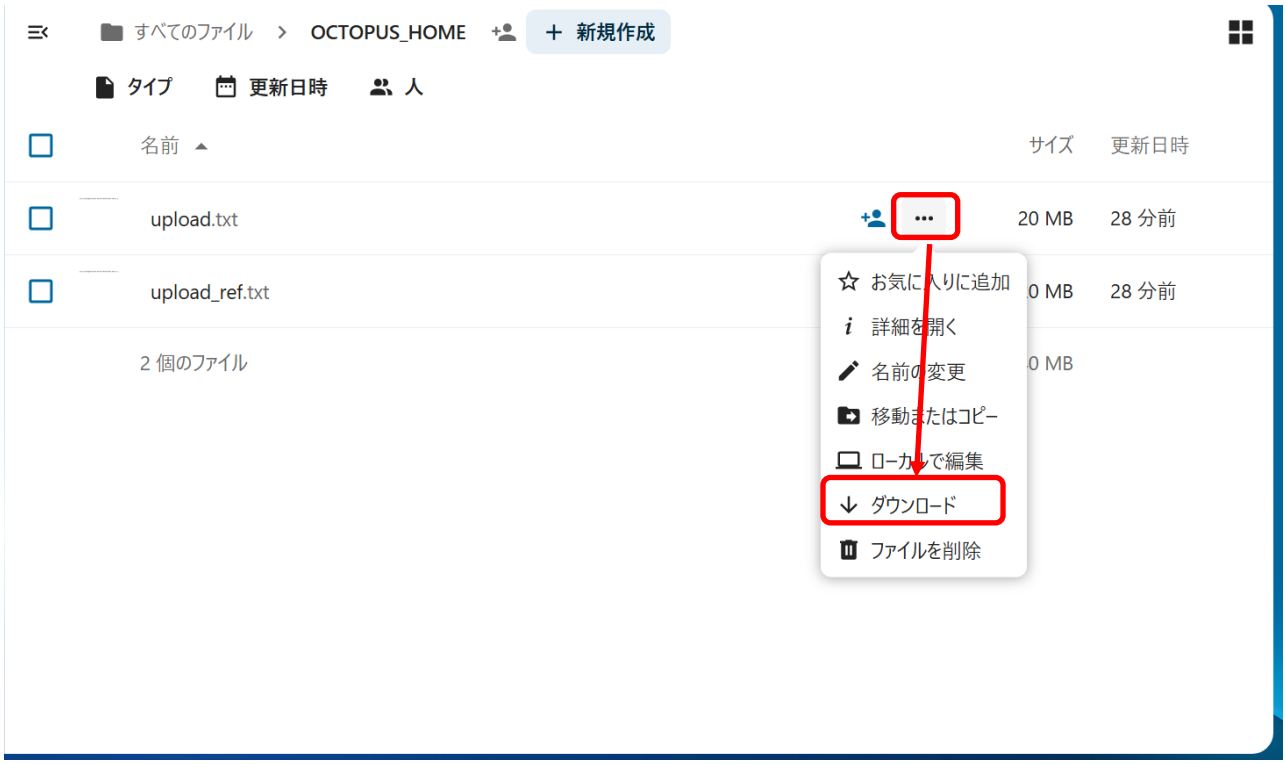
➢ **ダウンロード方法**

(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「OCTOPUS_HOME」内の「upload.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.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。

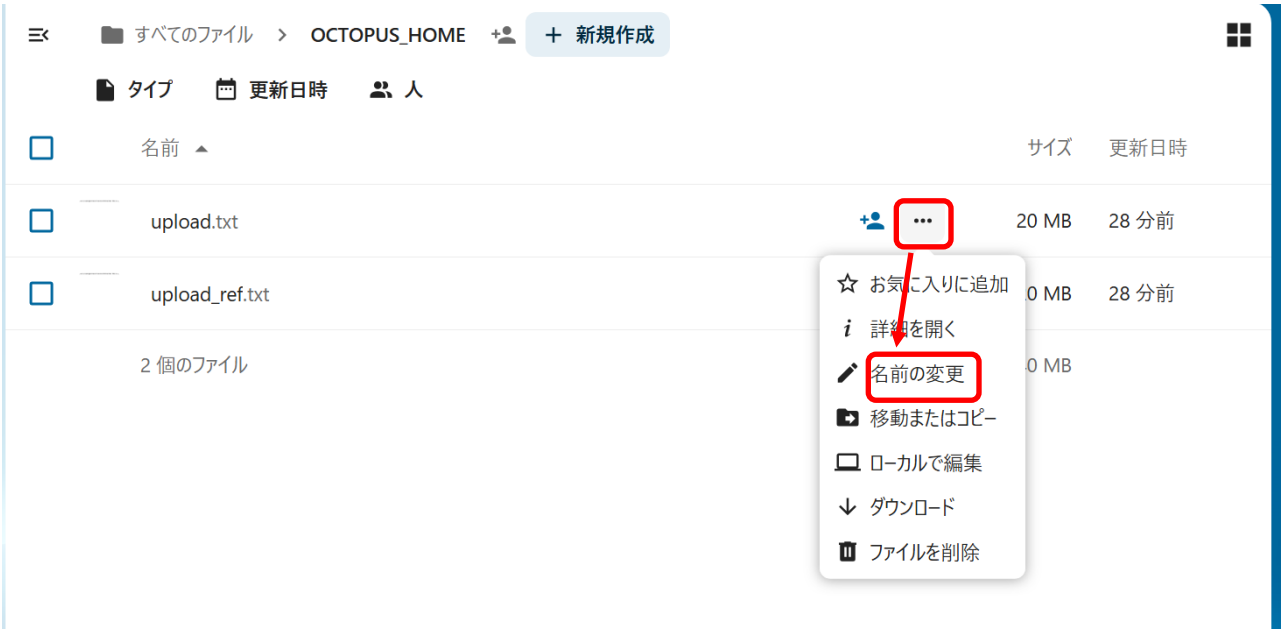
ダウンロードファイルの保存先指定は使用している Web ブラウザの設定に依存します。

- **フォルダ名・ファイル名変更**

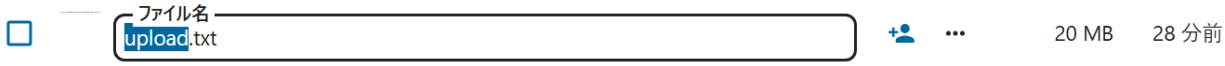
フォルダ名・ファイル名の変更は以下の手順になります。
以下の例では「upload.txt」の名前を変更します。

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。

OCTOPUS ユーザホーム領域

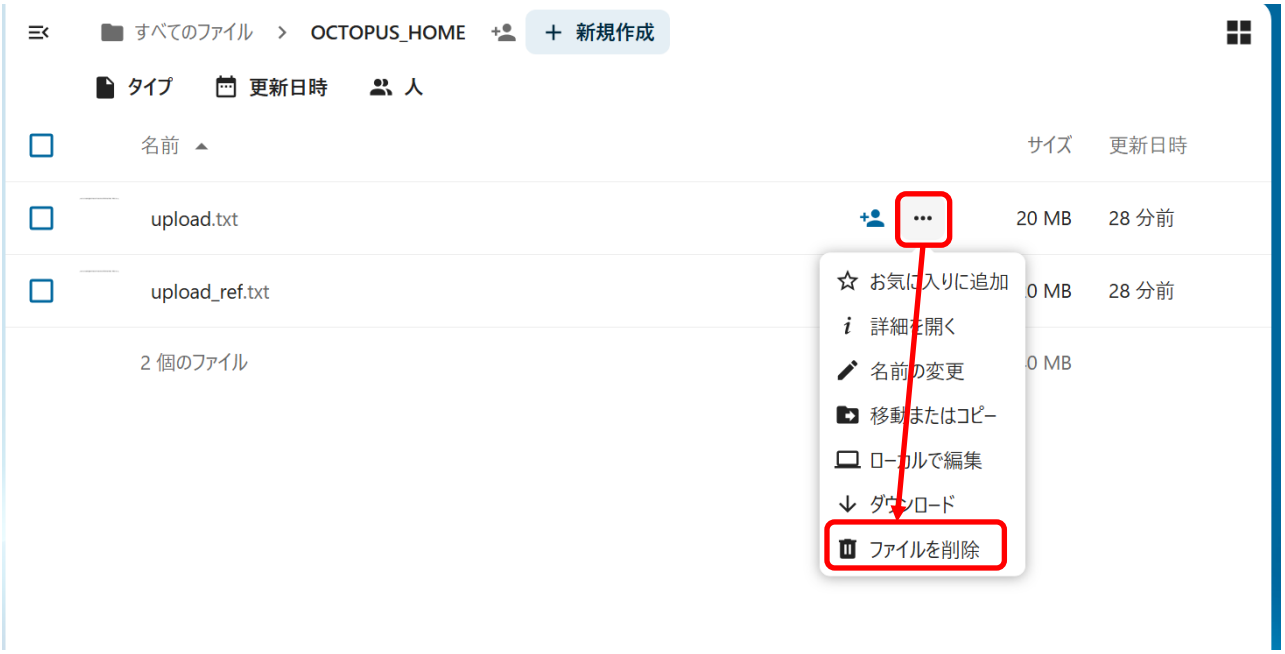


対象のフォルダ名・ファイル名を編集できるようになりますので任意の名前に変更し、エンターキーを入力します。



➢ **フォルダ・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「upload.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「フォルダを削除」、「ファイルを削除」をクリックします。



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。

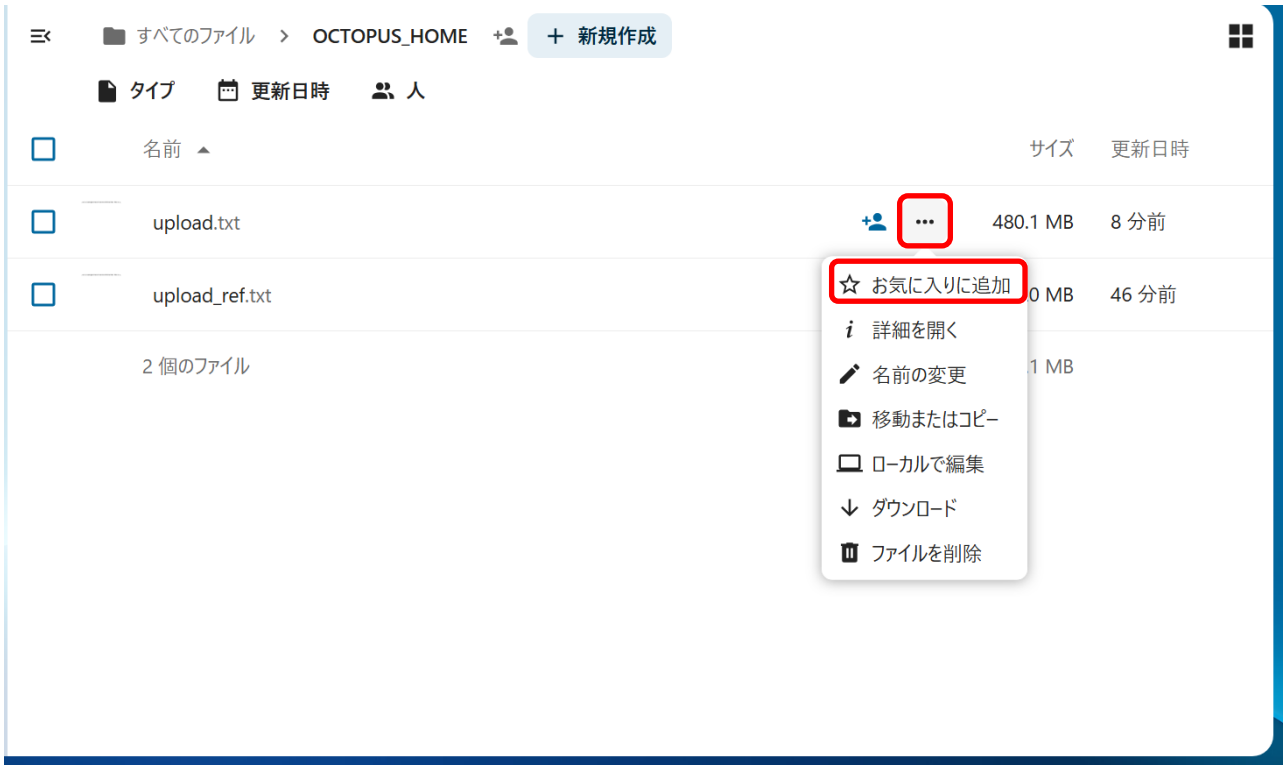
➢ **お気に入り**

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.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。

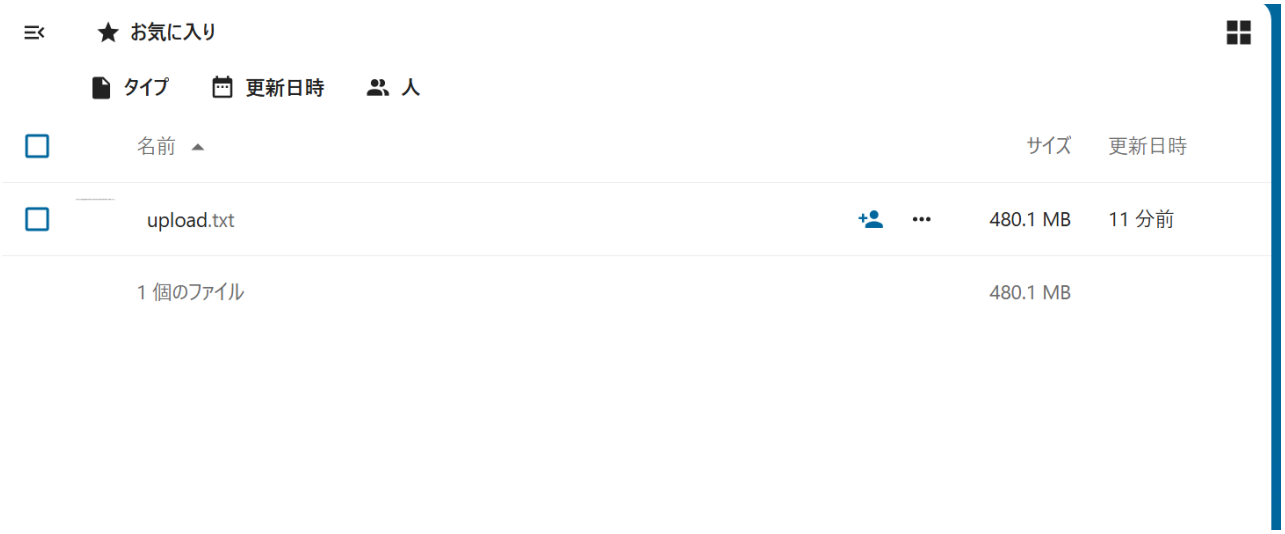
現在、お気に入りに登録しているフォルダ・ファイルを参照するには左メニューの「お気に入り」をクリックし、参照したいファイルをクリックします。

(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.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。

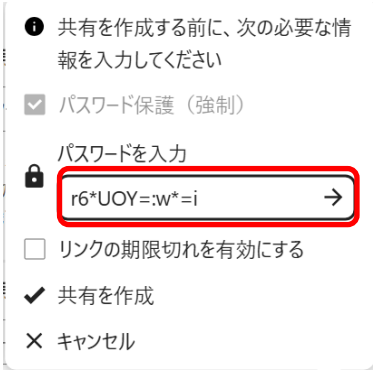
(1) フォルダ・ファイル共有設定（URL 共有）

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.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「公開リンクを作成」の右側「＋」をクリックします。

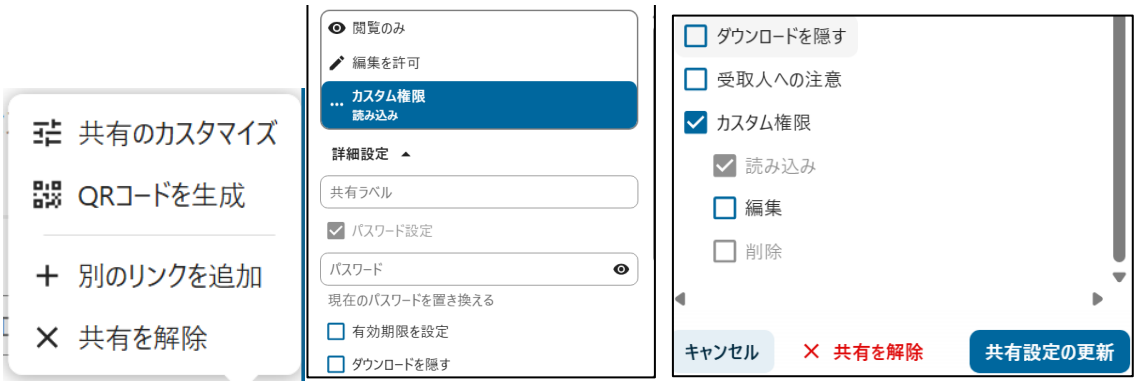
以下の画面が表示されますので「パスワードを入力」に表示されるパスワードを控えるか、任意のパスワードを設定します。任意のパスワードを設定する場合は 12 文字以上にする必要があります。その後、パスワード右横の「→」をクリックします。



「公開リンクを作成」と表示されていた部分が、「URL で共有」と切り替わり、右側のアイコンが以下のように変わりますので、「…」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。

以下、「共有のカスタマイズ」をクリックした時の画面と説明です。



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

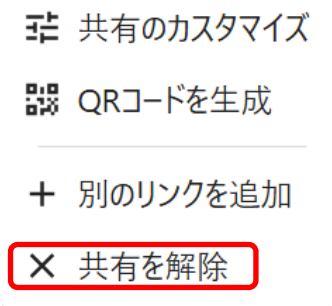
フォルダ・ファイルの共有解除は以下の手順になります。
以下の例では「upload.txt」を共有解除します。

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。



以下のように「URL で共有」と表示されていた部分が、「公開リンクを作成」と切り替われば共有解除完了です。

7.1.4 外部ストレージの追加

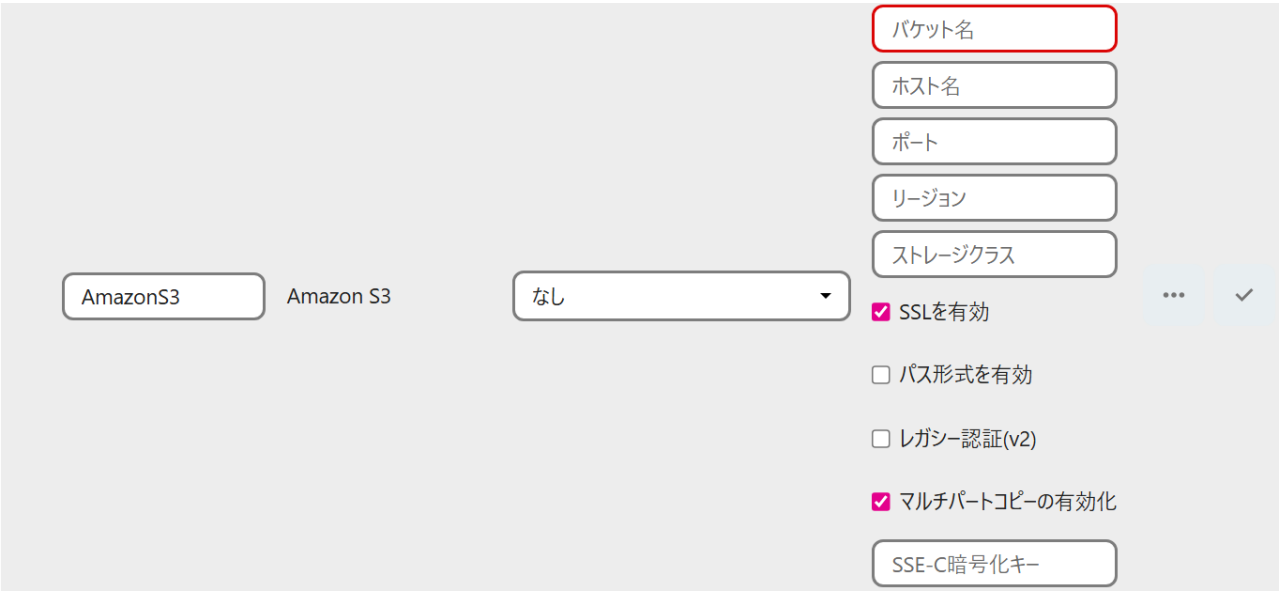
外部ストレージを追加することで Amazon S3 API に対応したストレージを、Nextcloud の環境から利用することが可能です。

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。

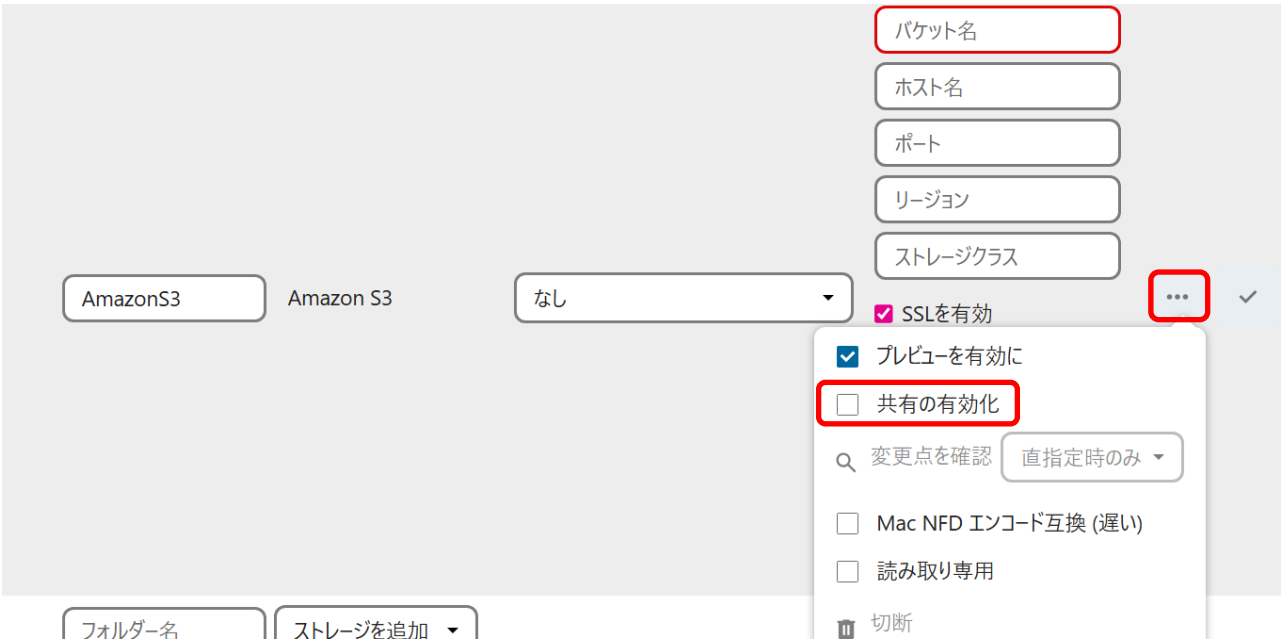
左メニューの「外部ストレージ」をクリックします。

「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。

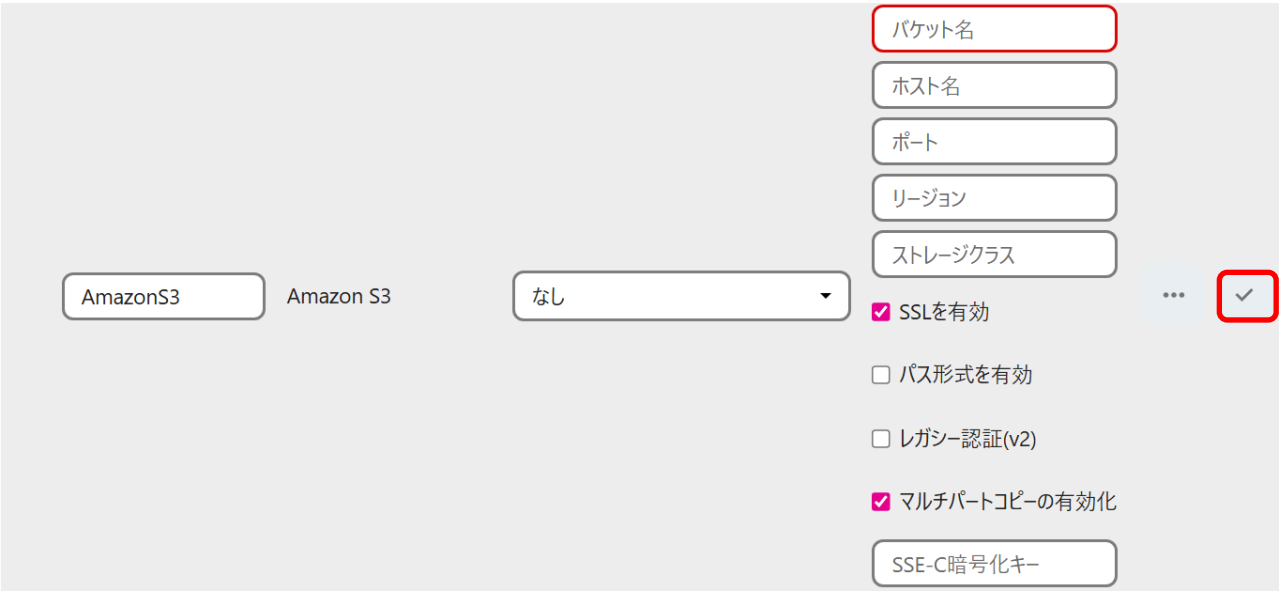
以下の設定項目が表示されますので、必要情報を入力します。



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

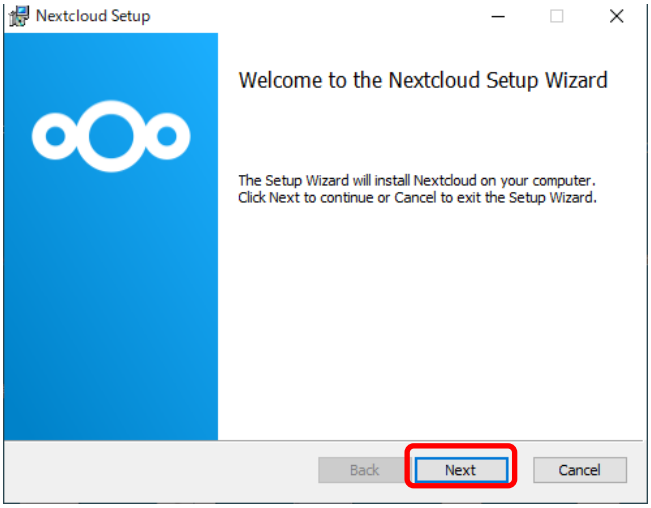
Nextcloud 環境は、デスクトップクライアントアプリが用意されており、クライアントアプリ経由で利用することも可能です。クライアントアプリのインストール作業には再起動を伴います。インストールを行う際は編集中のファイルなど保存して実行してください。

(1) インストール

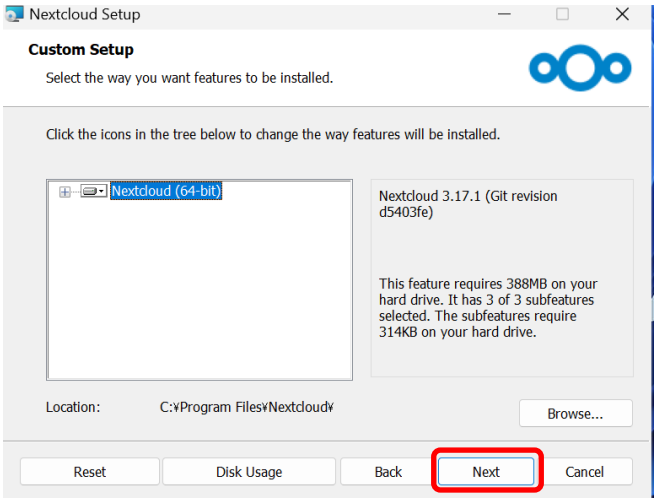
以下の URL からインストーラをダウンロードします。

https://nextcloud.com/install/#install-clients

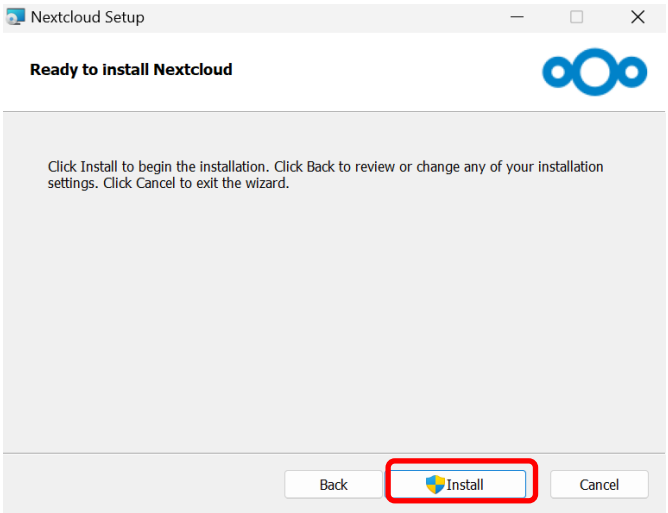
ダウンロードしたインストーラを実行します。
インストールが開始されますので「Next」をクリックします。



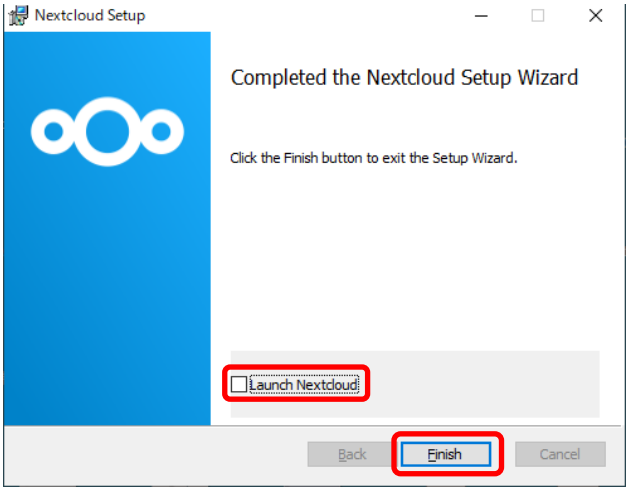
設定変更は特に必要ないため「Next」をクリックします。



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。



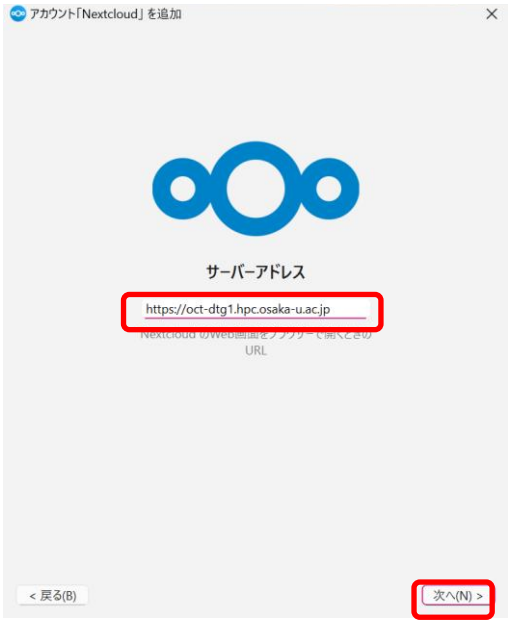
(2) 設定

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「ログイン」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。

https://oct-dtg1.hpc.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。

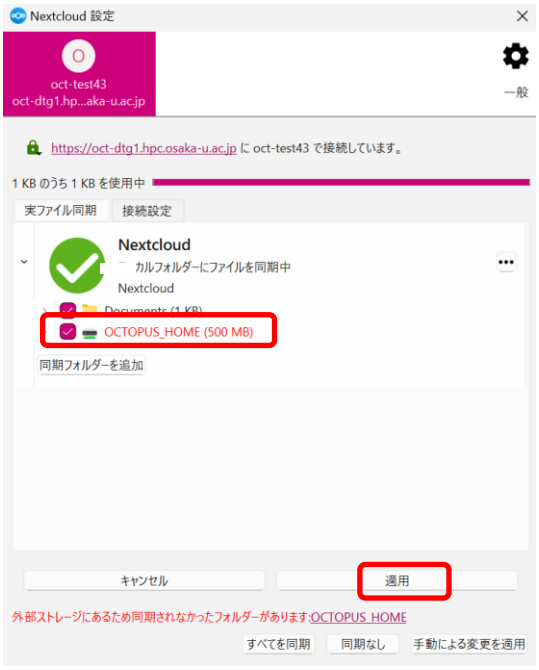
以下の画面が表示されますので、「アクセスを許可」をクリックします。

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。

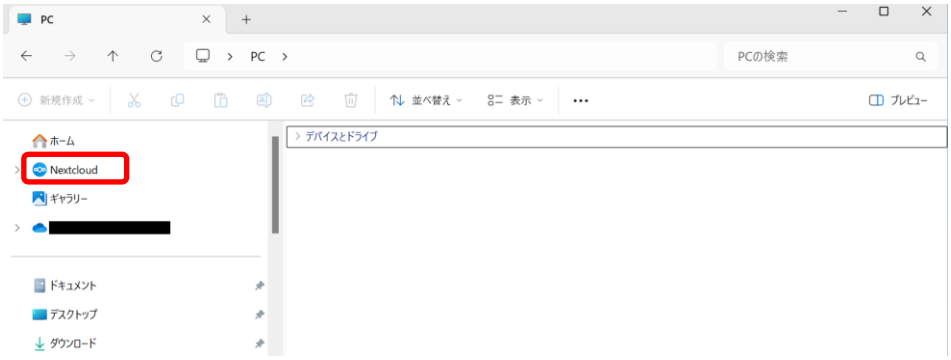
アプリケーションに戻り、以下の画面が表示されますので、「指定された容量以上のフォルダは同期前に確認」のチェックを外し、「接続」をクリックします。

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、「OCTOPUS_HOME」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。

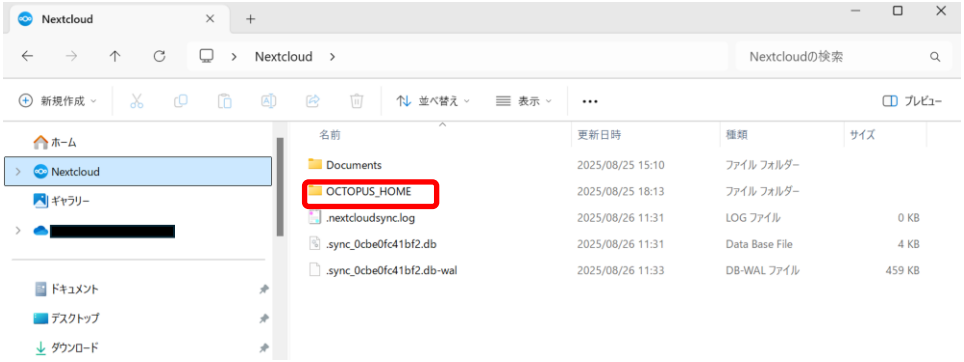


エクスプローラを開くと左側に「Nextcloud」が追加されていますのでクリックします。



「.」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。

「OCTOPUS_HOME」を開きます。ファイルの読み書きを行えます。



7.2 SQUID からのファイル転送

OCTOPUS と SQUID 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 SQUID も利用している方向けに、OCTOPUS⇔SQUID 間でのファイル転送手順を説明します。

OCTOPUS と SQUID 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。SQUID のファイ

ルシステムは、OCTOPUS 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	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

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

※OCTOPUS の計算ノードから SQUID 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 ~/
```

8 その他のサービス

8.1 統計情報用ポータルシステムの利用方法

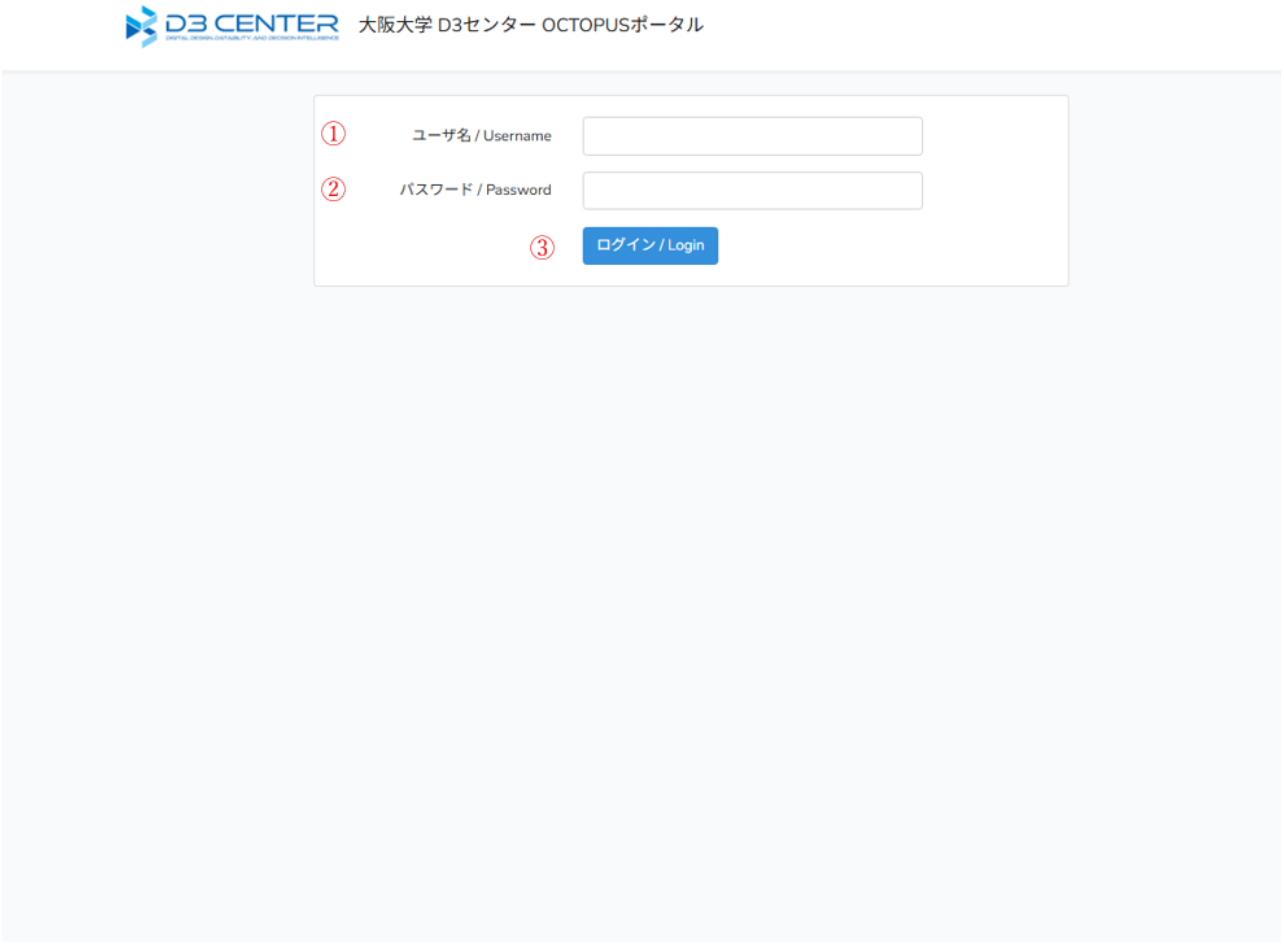
8.1.1 ポータルサイトログイン方法

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

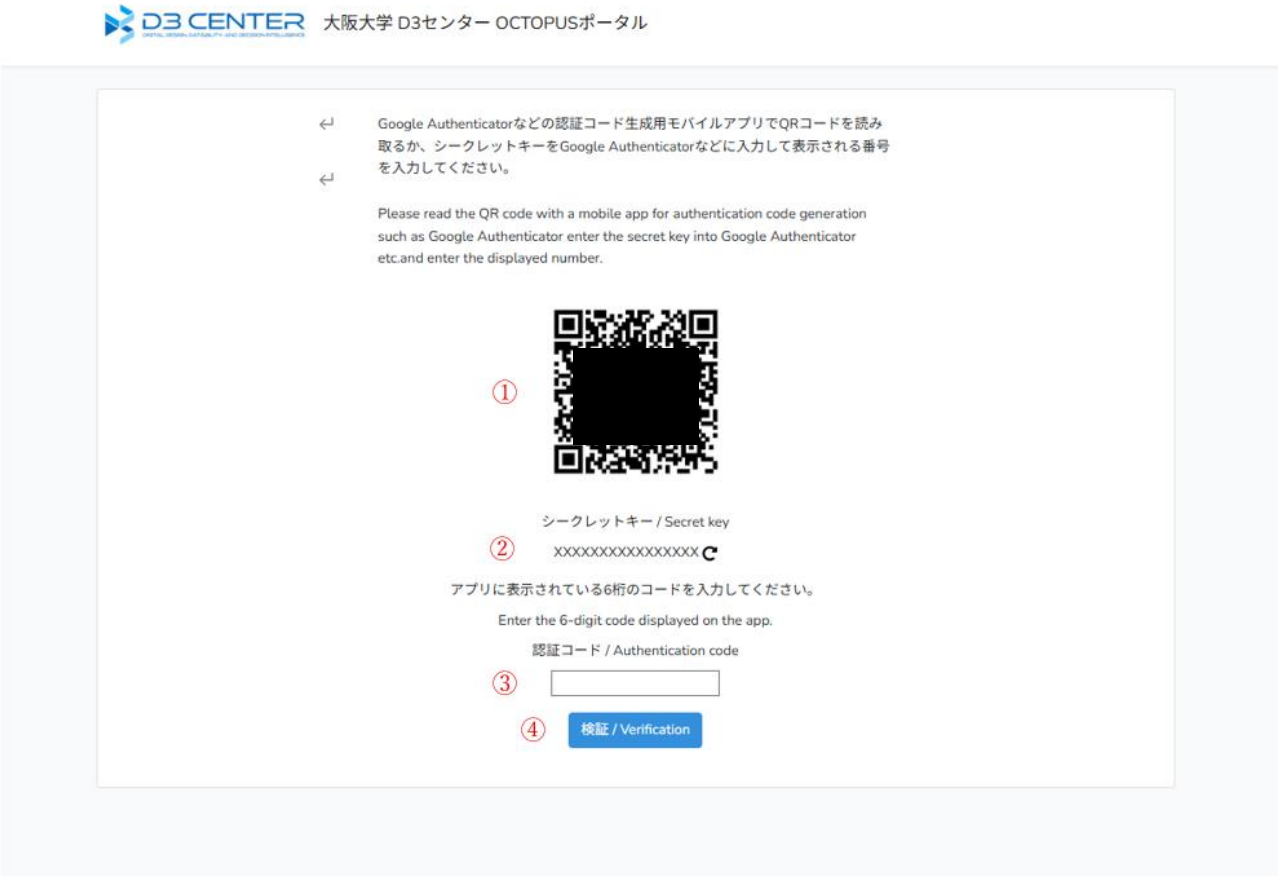
https://octopusportal.hpc.osaka-u.ac.jp/portal/login にアクセスするとポータルシステムのログイン画面が表示されます。



◆　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。



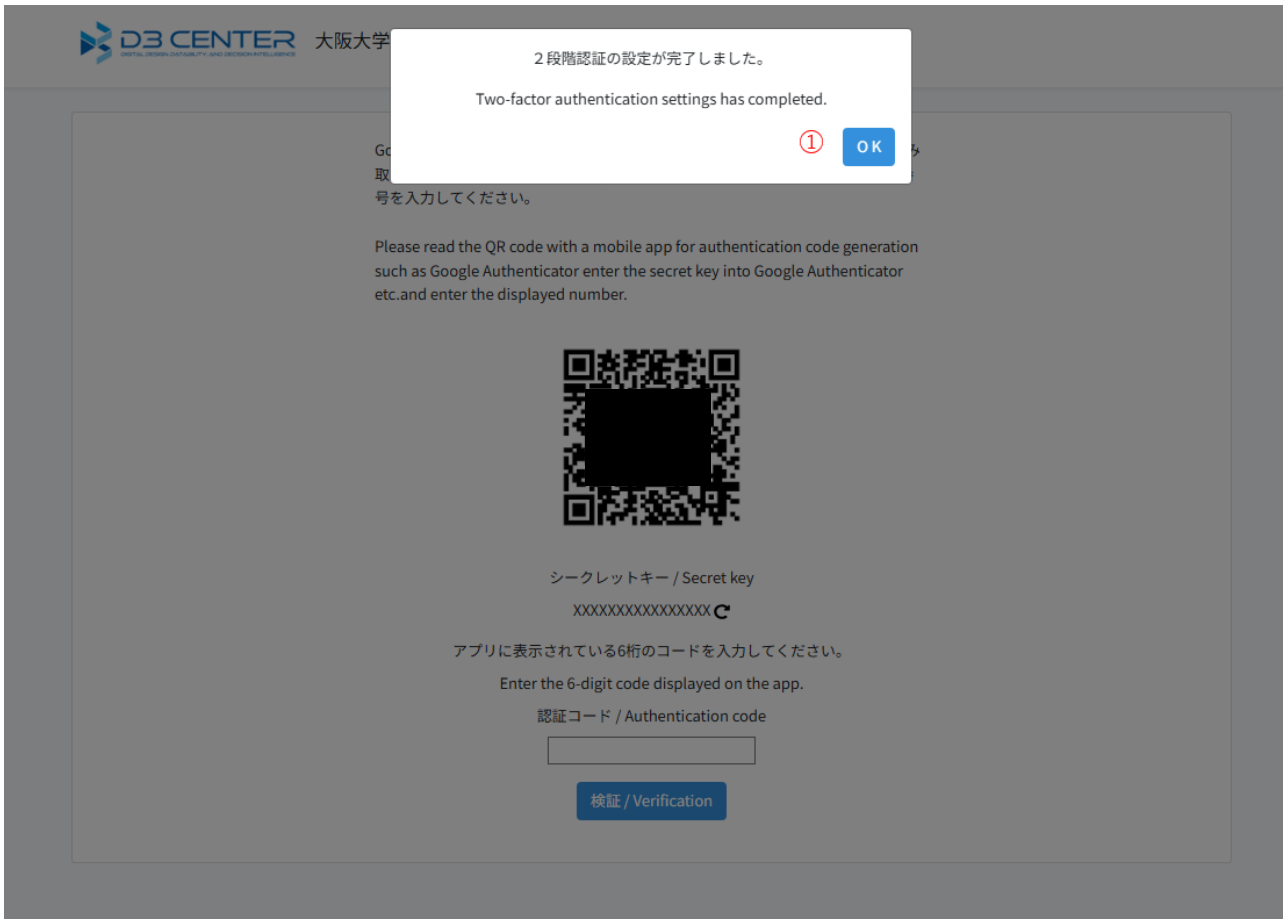
◆　2 段階認証設定手順
①　2 段階認証アプリケーションで表示されている QR コードを読み取ってください。
②　QR コードの読み取りができない場合は、シークレットキーを 2 段階認証アプリケーションに入力してください。
③　2 段階認証アプリケーションに表示されている【認証コード】を「認証コード」に入力してください。
④　「検証」ボタンをクリックしてください。

◆　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。

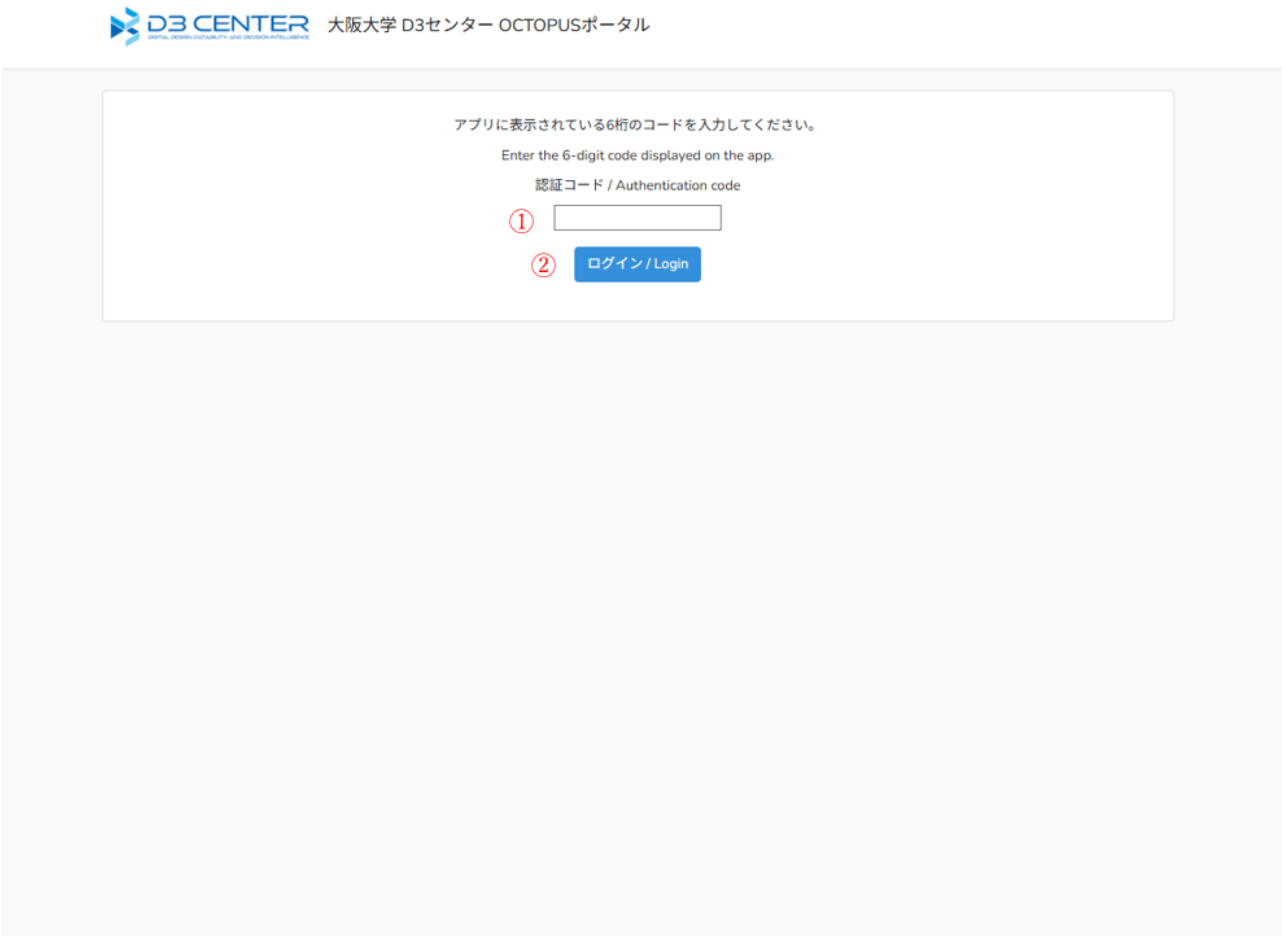
①　OK ボタン

「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。



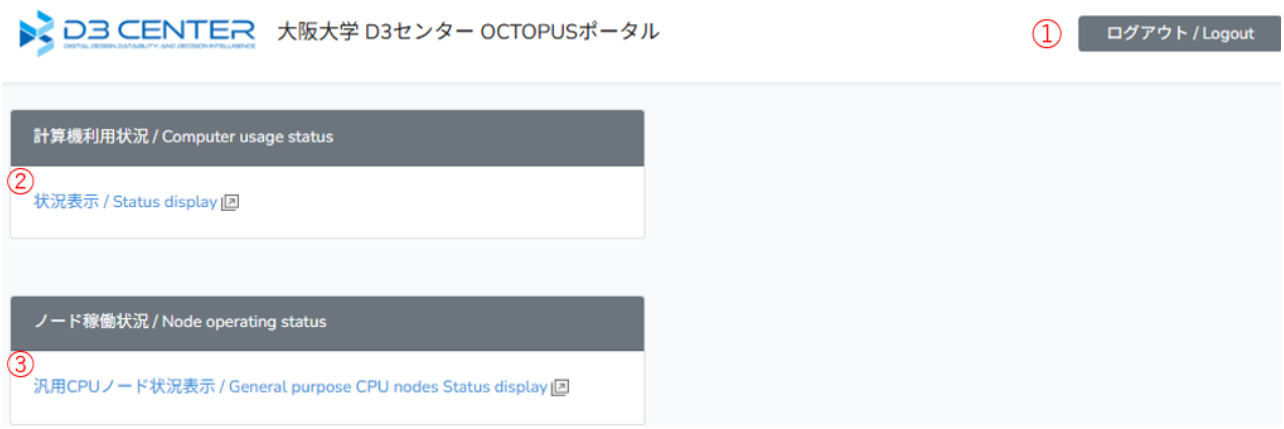
◆ 2 段階認証手順

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。



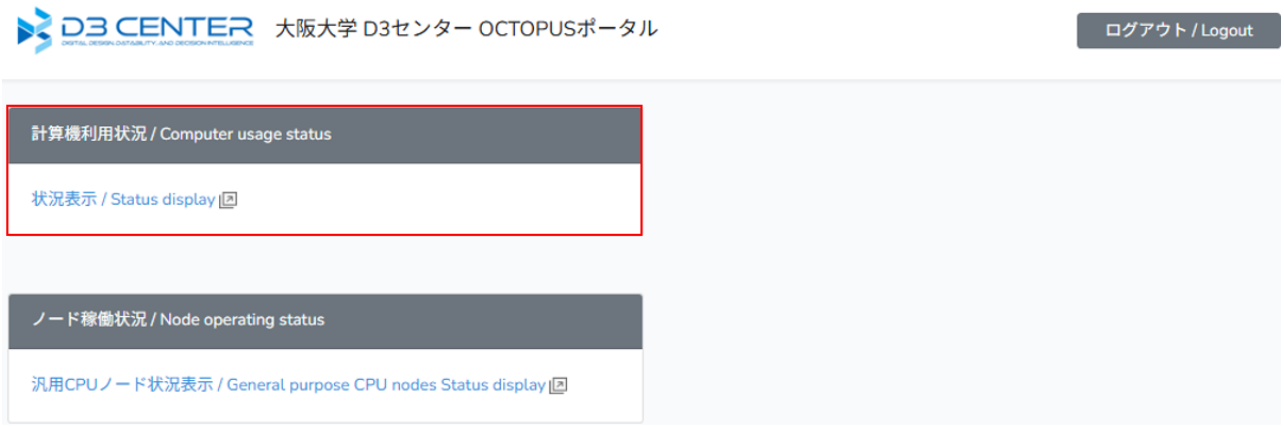
① ログアウトボタン
「ログアウト」ボタンをクリックすると、ログアウト処理を行いログイン画面に遷移します。

② 計算機利用状況 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
「汎用 CPU ノード状況表示」リンクを選択すると、新しいタブ画面にノード稼働状況を表示します。

8.1.2 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

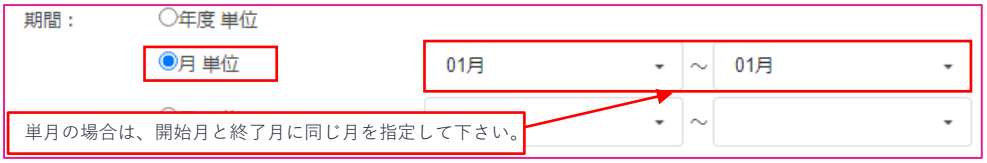
①　年度を指定する

利用状況を表示する年度を選択します。2025 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 2026/4/1 であれば 2026 年度から選択可能です。
　登録日が 2026/3/31 であれば 2025 年度から選択可能です。

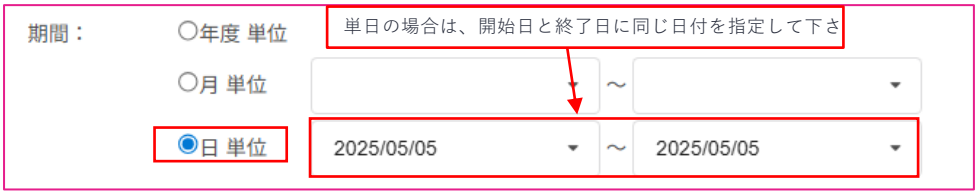
②　期間を指定する

・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。

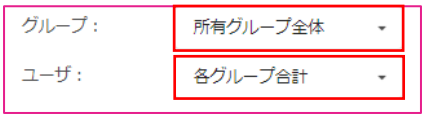


③　グループ・利用者を指定する

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。

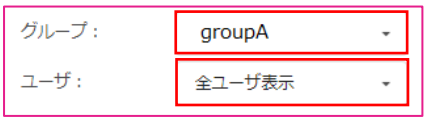
A)　申請代表者

・　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

・　特定のグループと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　検索する

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

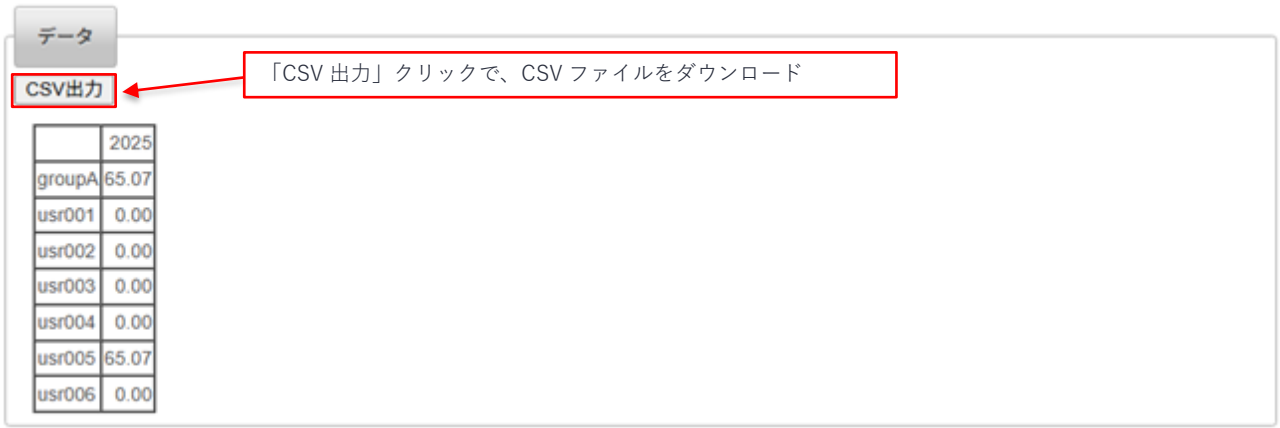
ディスク使用量(ホーム、拡張領域)

(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。



	2025
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」と同じ内容を CSV 形式でダウンロードします。

	A	B
1		2025
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

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。

	2024/04	2024/05	2024/06	2024/07	2024/08	2024/09	2024/10	2024/11	2024/12	2025/01	2025/02
usr001	0.00	0.00	0.00	8.00	8.00	8.00	8.00	8.00	8.00	21.00	22.00

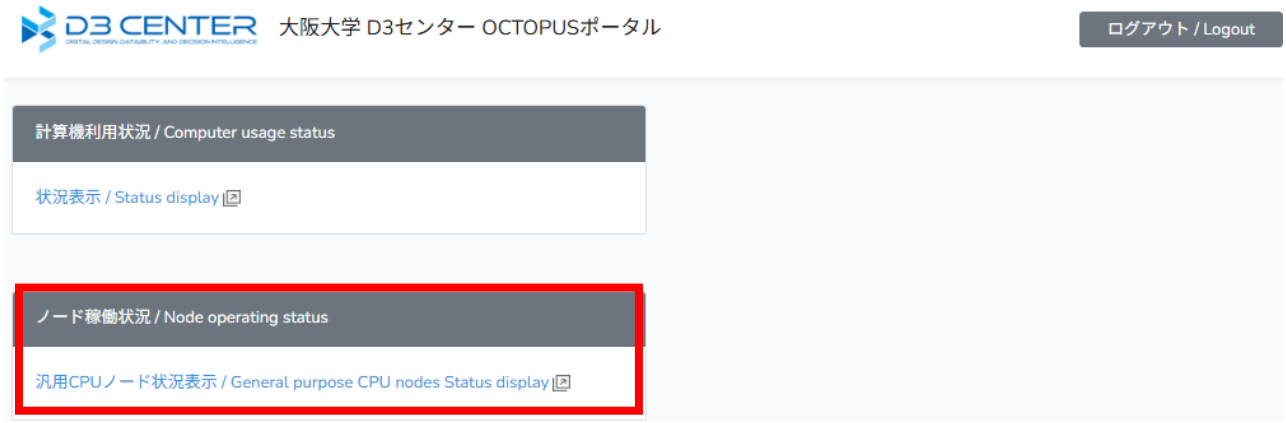
④　月単位・日単位の CSV ファイルイメージ

画面に表示している「データ」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		2024/04	2024/05	2024/06	2024/07	2024/08	2024/09	2024/10	2024/11	2024/12	2025/01	2025/02
2	usr001	0	0	0	8	8	8	8	8	8	21	22

※Excel での表示例

8.1.3 ノード稼働状況 Web 表示

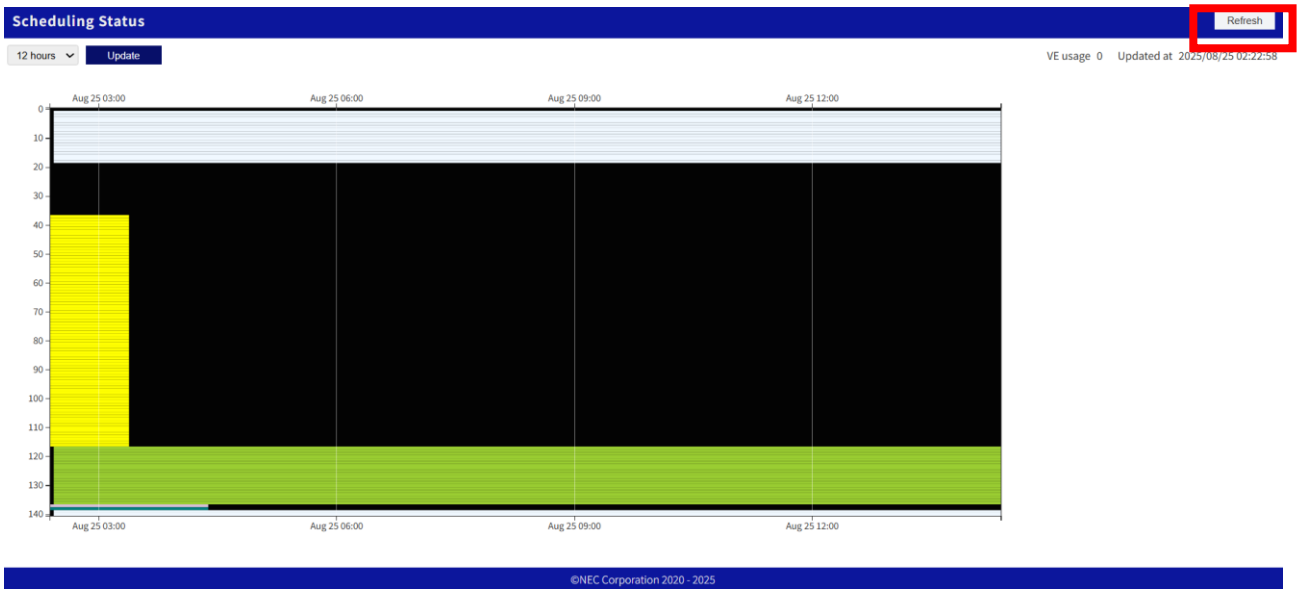


「汎用 CPU ノード状況表示/General purpose CPU nodes Status display」のリンクを選択します。

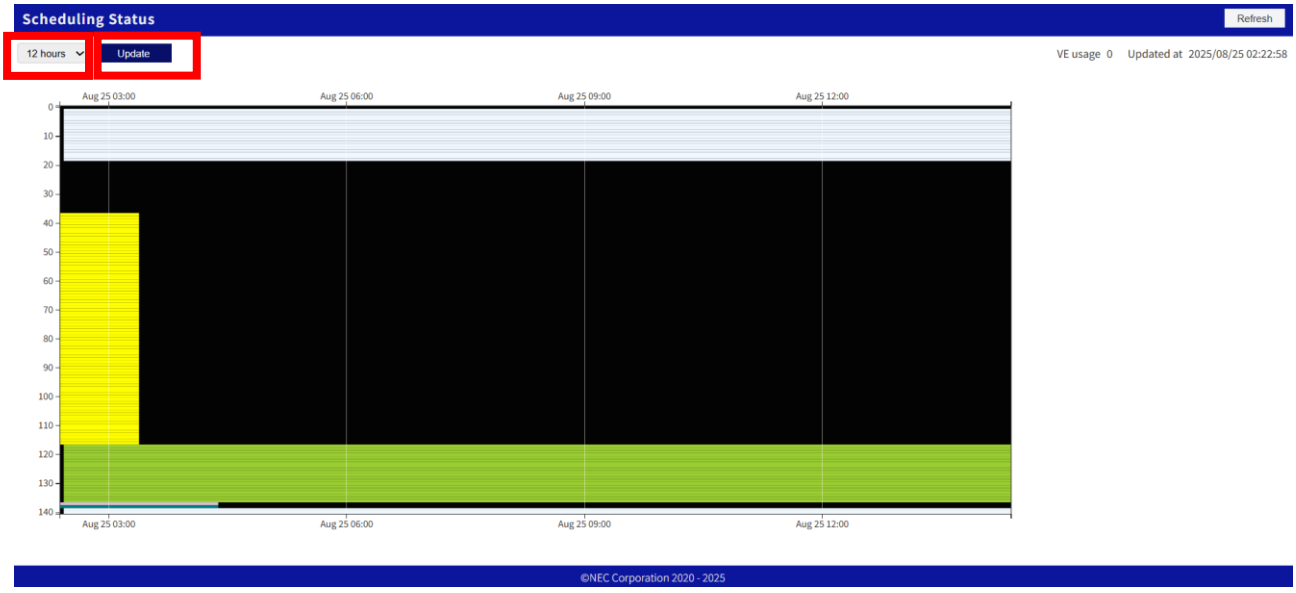
「汎用 CPU ノード状況表示/General purpose CPU nodes Status display」に接続後、以下の画面が表示されます。
縦軸はジョブサーバ番号、横軸は時間を示します。
ジョブサーバ番号とは CPU ノードに割り振られている一意の番号です。
HPC フロントエンドで「qstat -St」コマンドを実行すると確認できます。
グラフ内の横棒はリクエストを表し、リクエスト毎に色が割り当てられます。

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。

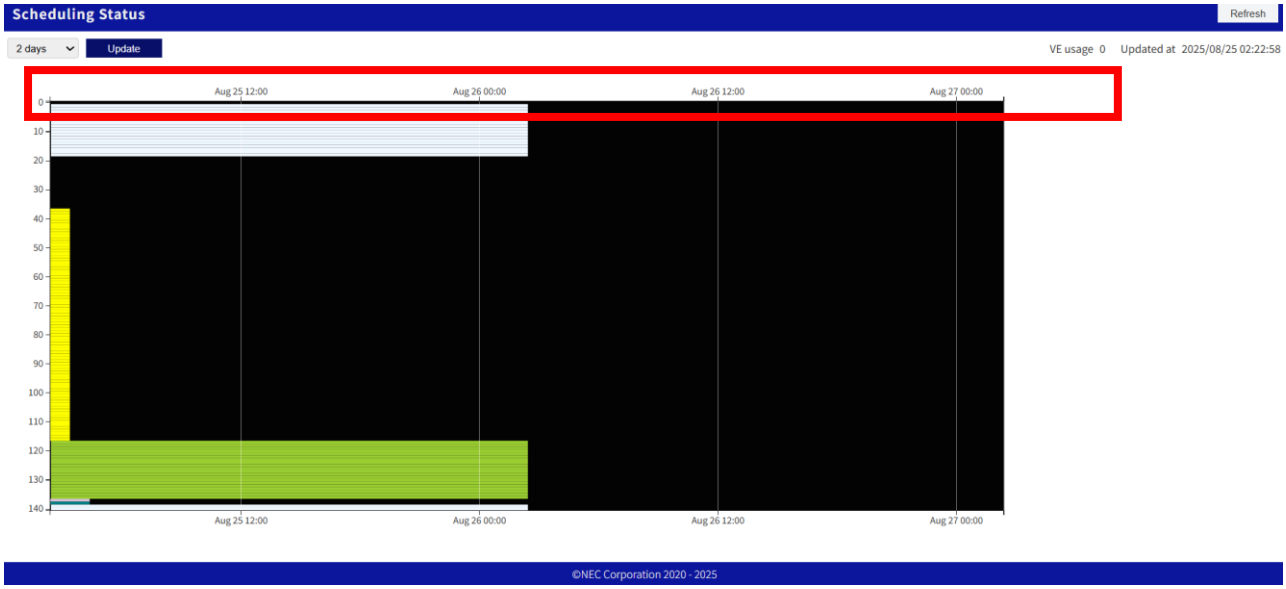
画面右上の「Refresh」ボタンを押下すると、最新のノード状況にページが更新されます。



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。


Scheduling Status
Refresh
2 days
Update
VE usage 0
Updated at 2025/08/25 02:22:58
Aug 25 12:00
Aug 26 00:00
Aug 26 12:00
Aug 27 00:00
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
Aug 25 12:00
Aug 26 00:00
Aug 26 12:00
Aug 27 00:00
©NEC Corporation 2020 - 2025

以上