NoSQL/SQLデュアルインターフェースを備えたIoT向けデータベースGridDB
～クラウドでGridDBを使ってみましょう～

TOSHIBA

東芝デジタルソリューションズ株式会社　GridDBコミュニティ版担当
野々村 克彦

2023.3.10, 11

Contents

01

GridDBの概要

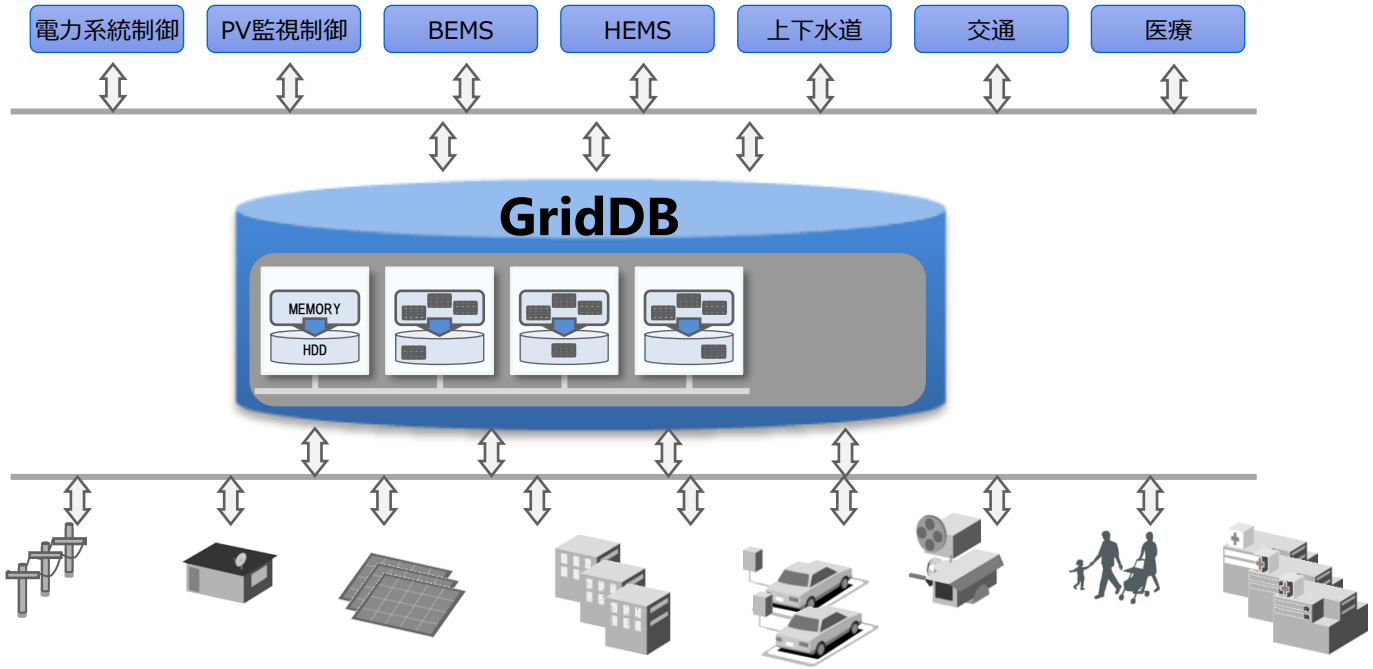
GridDBの概要

①GridDBとは？
②GridDBはオープンソース？
③オープンソース化の目的
④特徴

①GridDBとは？

- 日本発のビッグデータ／IoT向けデータベース

※IoT：モノのインターネット（Internet Of Things）。大量のモノ（センサなど）から得られるデータがインターネットにつながること。



②GridDBはオープンソース？

GridDB
Community Edition

高頻度・大量に発生する時系列データの蓄積とリアルタイムな活用をスムーズに実現する次世代の
オープンソースデータベース

GridDB
Enterprise Edition

高頻度・大量に発生する時系列データの蓄積とリアルタイムな活用をスムーズに実現し、ビジネスを大きく成長させるために
最適化された次世代のデータベース

GridDB Cloud

高頻度・大量に発生する時系列データの蓄積とリアルタイムな活用をスムーズに実現する
クラウドデータベースサービス

社会インフラ、製造業を中心に、高い信頼性・可用性が求められるシステムに適用されている

③GridDB　オープンソース化の目的

- ビッグデータ技術の普及促進
  - 多くの人に知ってもらいたい、使ってみてもらいたい。
  - いろんなニーズをつかみたい。
- 他のオープンソースソフトウェア、システムとの連携強化

- V2.8 (2016年)
  NoSQL機能をGitHub上にソース公開
  https://github.com/griddb/griddb_nosql
- V4.5 (2020年)
  SQL機能もソース公開
  https://github.com/griddb/griddb
- 最新版 V5.1 (2022年10月25日)

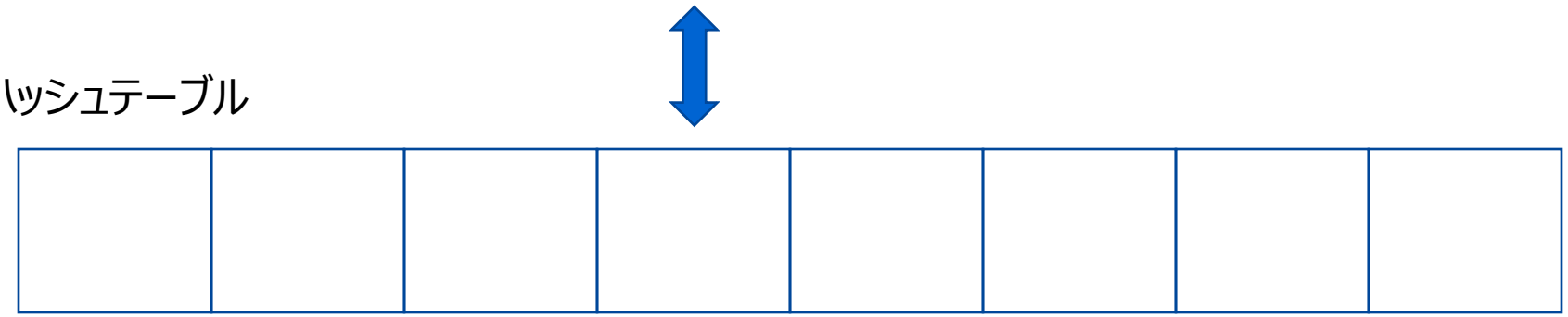
④GridDB CEの特徴

時系列データ指向	• データモデルはキー・コンテナ。コンテナ内でのデーター貫性を保証 • 巨大テーブルに対するインターバル（ハッシュ）パーティショニング • パーティショニング期限解放、分析関数(SQL)
開発の俊敏性と使いやすさ	• NoSQL（キーバリュー型）インタフェースだけではなく、SQLインタフェースを提供（デュアルインタフェース） • （SQLインタフェース）ジョインなど複数テーブルに対するSQL
高い処理能力	• メモリを主、ストレージを従としたハイブリッド型インメモリDB • （SQLインタフェース）SQLにおける分散並列処理 • （NoSQLインタフェース）バッチ処理 MultiPut/MultiGet/MultiQuery
拡張性	• ペタバイト級の大規模データへの対応 • コアスケールへの対応

※　チェックポイント、Redoログによる耐障害性への対応

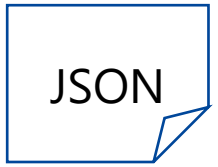
NoSQL DB (Key Value Store(KVS))とキー・コンテナモデル

<キー、バリュー>

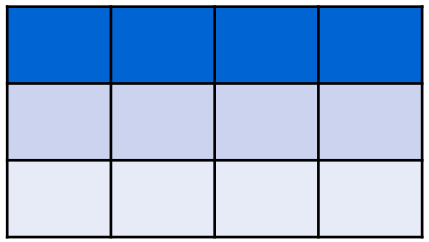


123

単純値：（例）Redis

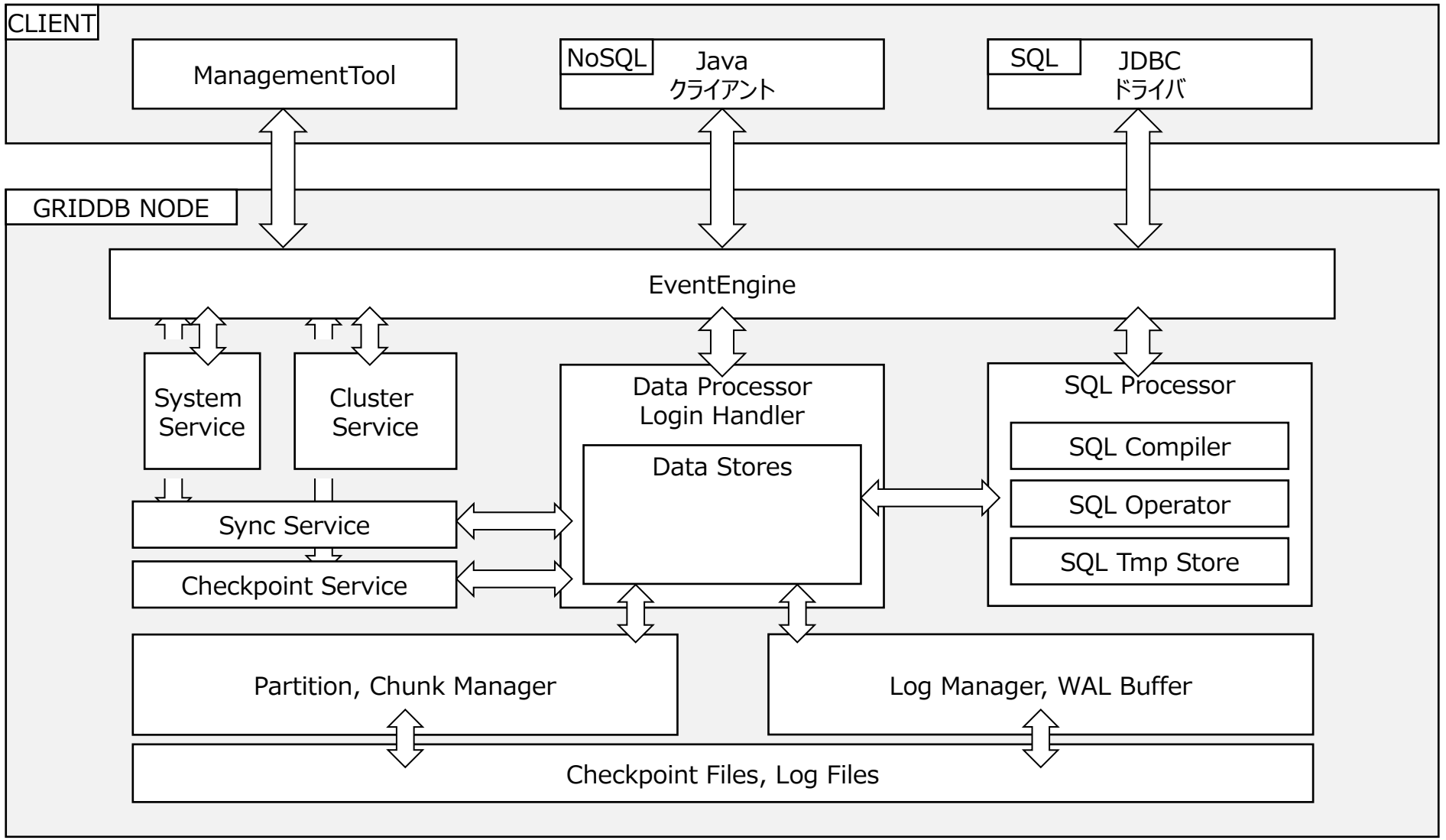


ドキュメント：（例）MongoDB

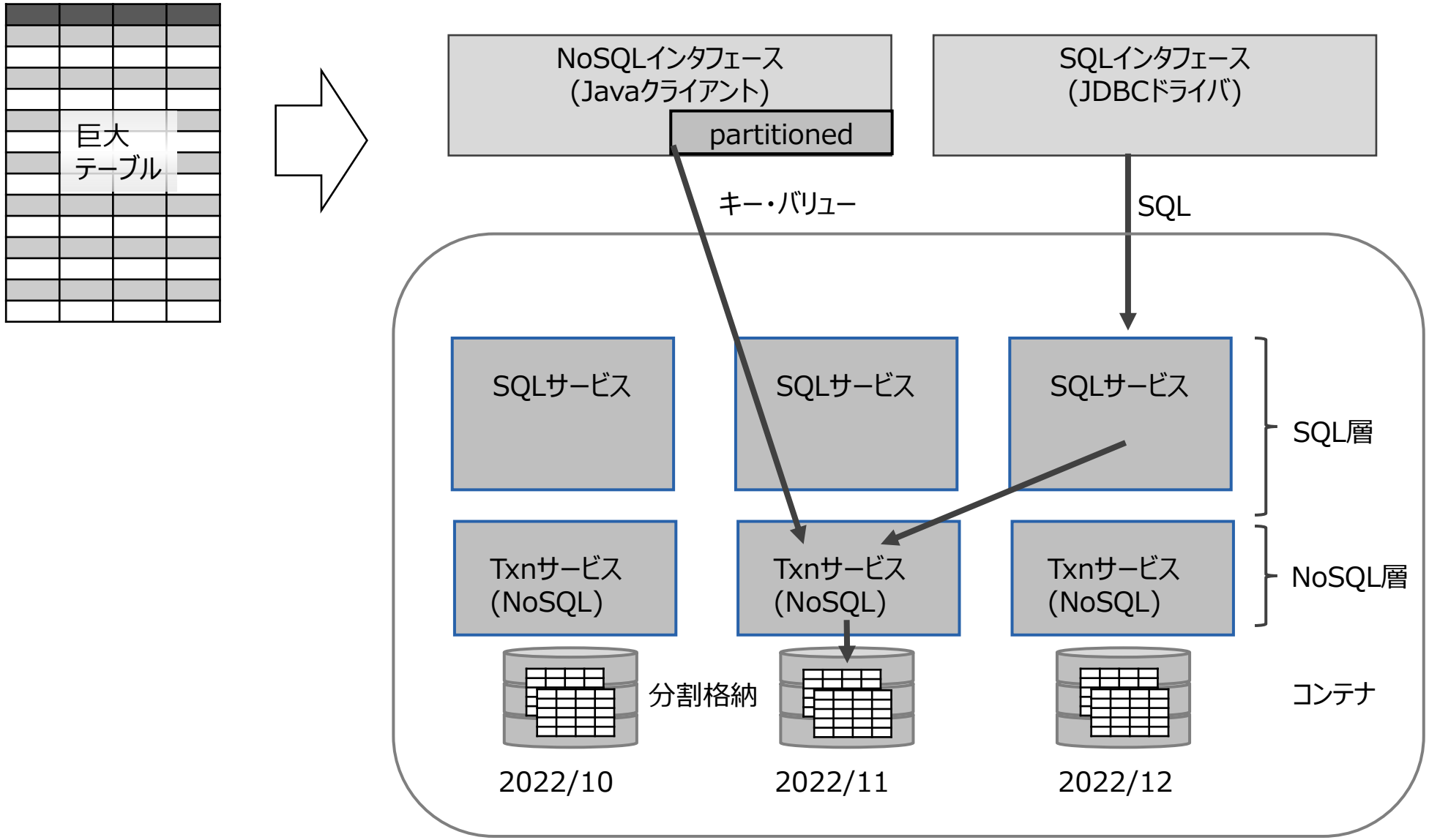


コンテナ（テーブル）： GridDB
※コンテナ（テーブル）名がキーになる
※索引、検索言語TQL、トランザクションをサポート

内部モジュール構成



デュアルインタフェースとテーブルパーティショニング



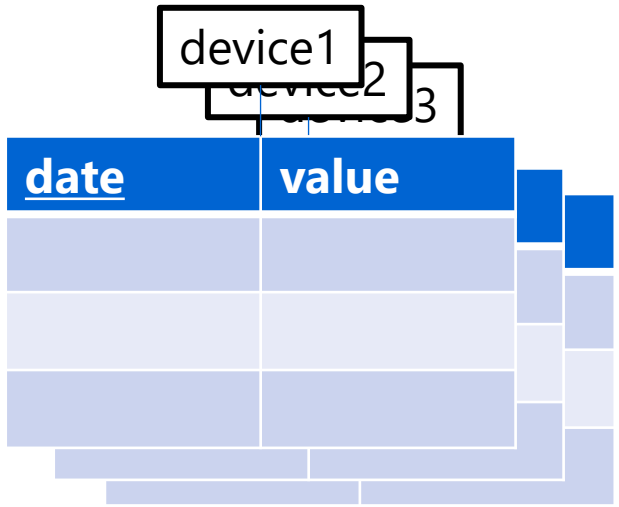
テーブルパーティショニング

- データ登録数が多い巨大なテーブルのデータを分散配置することで、プロセッサの並列実行を可能とし、巨大テーブルのアクセスを高速化するための機能
  - ハッシュパーティショニング
    - ✓ 選択基準：散らすべきキーにランダム性が高く、キーの間に処理上の関連性が無い場合
  - インターバルパーティショニング
    - ✓ 選択基準：散らすべきキーの数値的な範囲で散らしたい場合
  - インターバルハッシュパーティショニング
    - ✓ 選択基準：インターバルパーティショニングでは力不足の場合

```
-- ハッシュ
CREATE TABLE a3 (code INT, ts TIMESTAMP, dest STRING NOT NULL)
    PARTITION BY HASH(dest) PARTITIONS 10
-- インターバル
CREATE TABLE a1 (code INT, ts TIMESTAMP NOT NULL, dest STRING)
    PARTITION BY RANGE(ts) EVERY(1,DAY)
-- インターバルハッシュ
CREATE TABLE a4 (code INT NOT NULL, ts TIMESTAMP, dest STRING)
    PARTITION BY RANGE(ts) EVERY(1,DAY)
    SUBPARTITION BY HASH(dest) SUBPARTITIONS 2
```

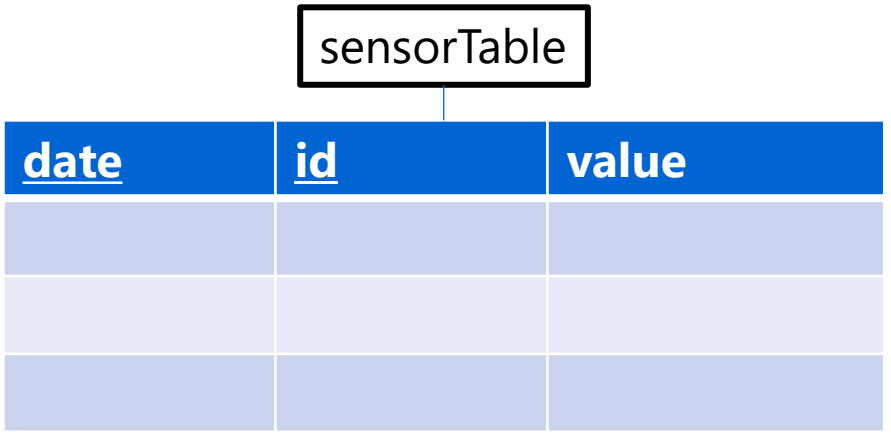
時系列データのスキーマ例

装置ごとに＜日時、センサ値＞のコンテナ



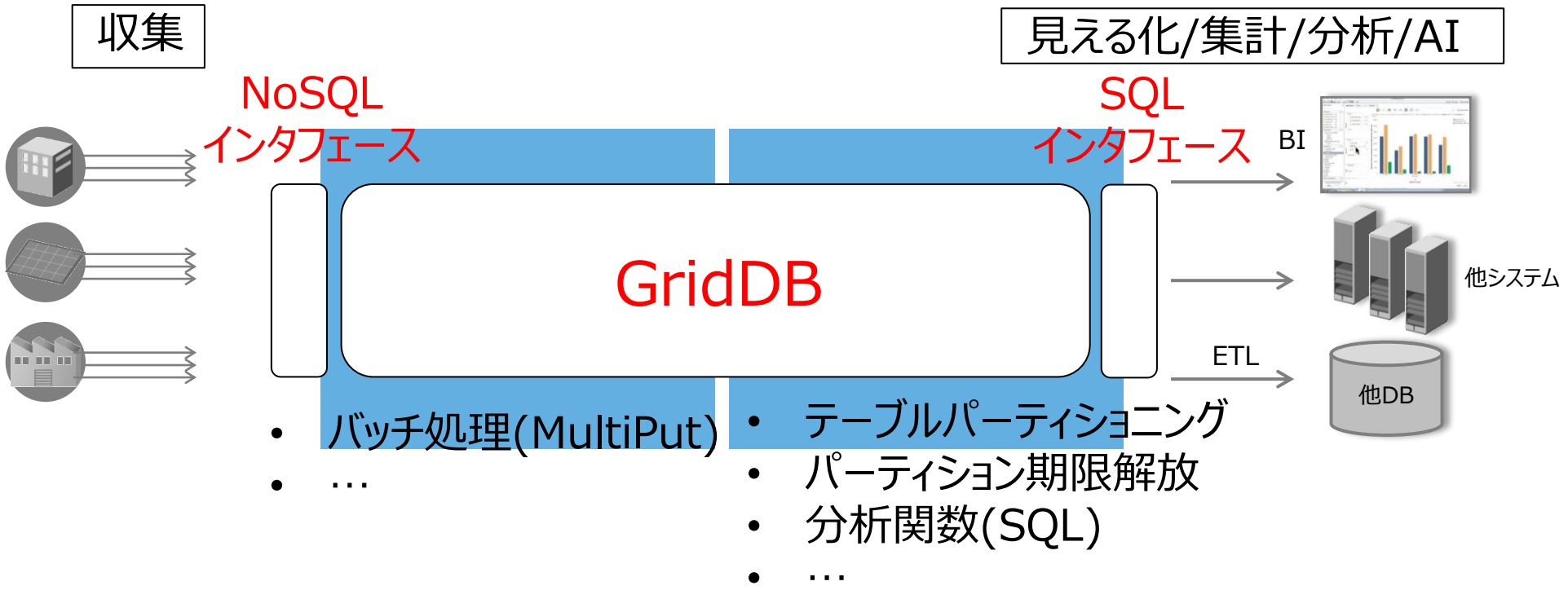

```
CREATE TABLE device1 (
 date TIMESTAMP, -- 日時
 value DOUBLE, -- センサ値);
```

＜日時、装置ID、センサ値＞のテーブル＋
インターバル（ハッシュ）パーティショニング




```
CREATE TABLE sensorTable (
  date TIMESTAMP, -- 日時
  id INTEGER, -- 装置ID
  value DOUBLE, -- センサ値
  PRIMARY KEY(date, id)
) PARTITION BY RANGE (date) EVERY (30, DAY);
SUBPARTITION BY HASH(id) SUBPARTITIONS 6;
-- 分割幅30日、サブパーティション数6の
     インターバルハッシュパーティショニング
```

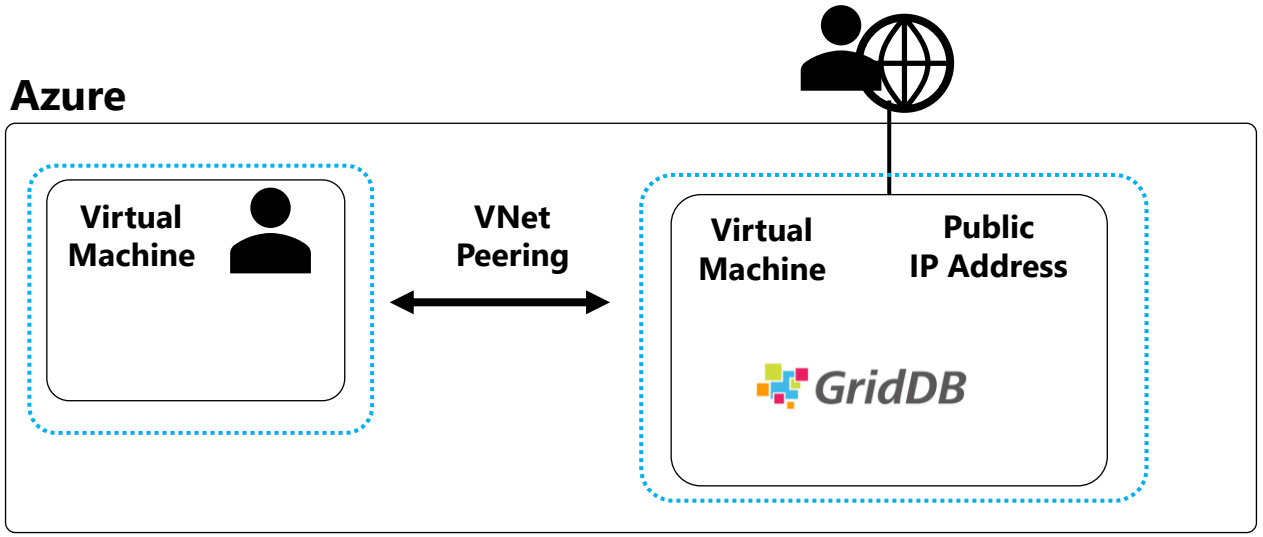
NoSQL/SQLデュアルインタフェースによるシステム化



- NoSQL＋SQLによる高速処理
- SQLインタフェースによる他システム連携強化

クラウド向け機能

- クラスタ構成とAPIからの接続
  - マルチキャスト方式
  - 固定リスト方式：GridDBサーバのIPアドレスを直接指定する方法
- APIの複数経路からの接続
  - 内部経路通信（クラスタノード間の通信経路と共通）
  - 外部経路通信

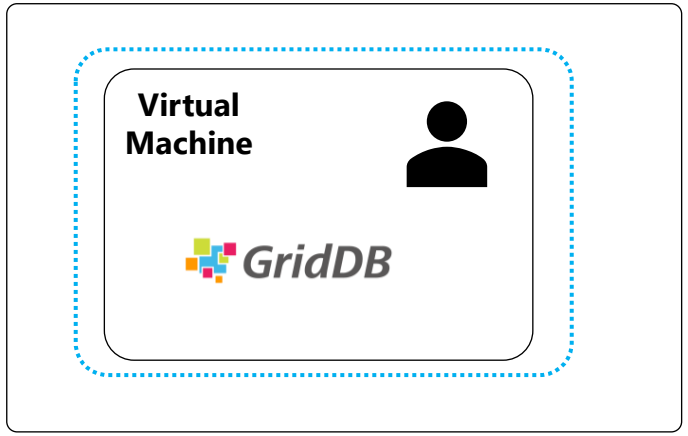


02

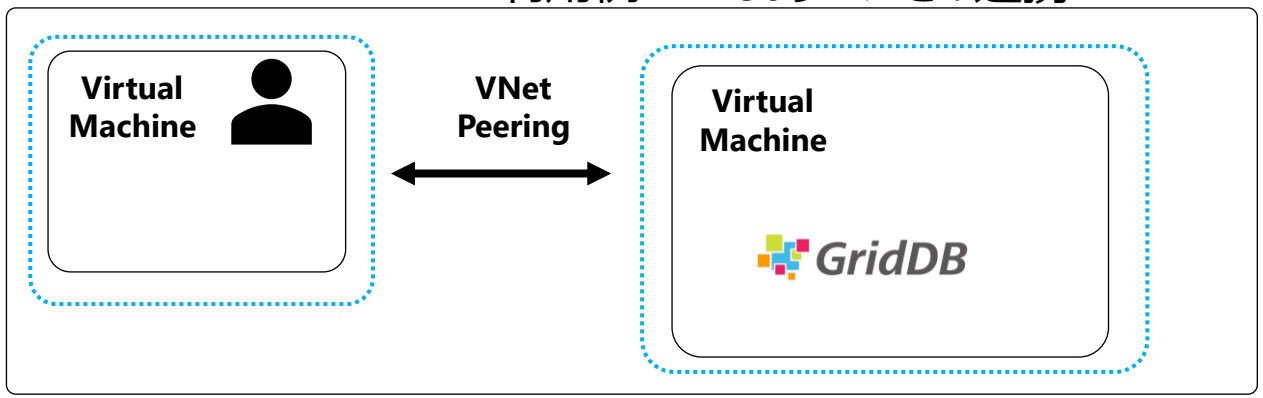
クラウド（Azure）でのGridDBの利用方法

Azure上のGridDB利用形態

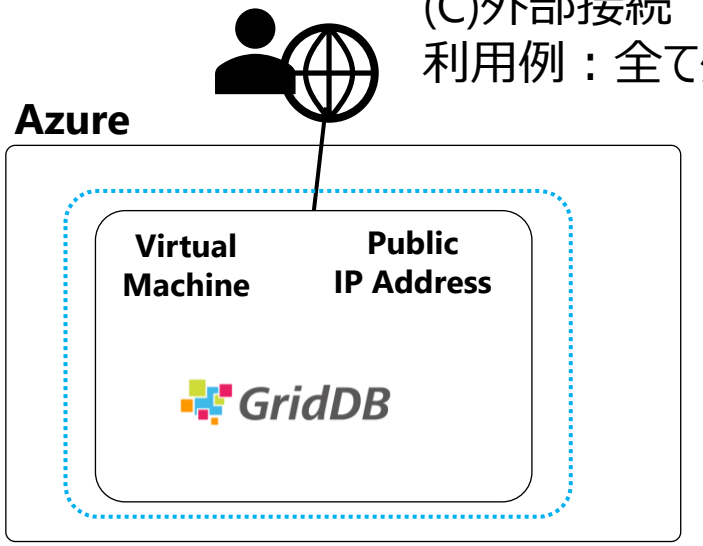
(A)ローカルアクセス
利用例：動作確認



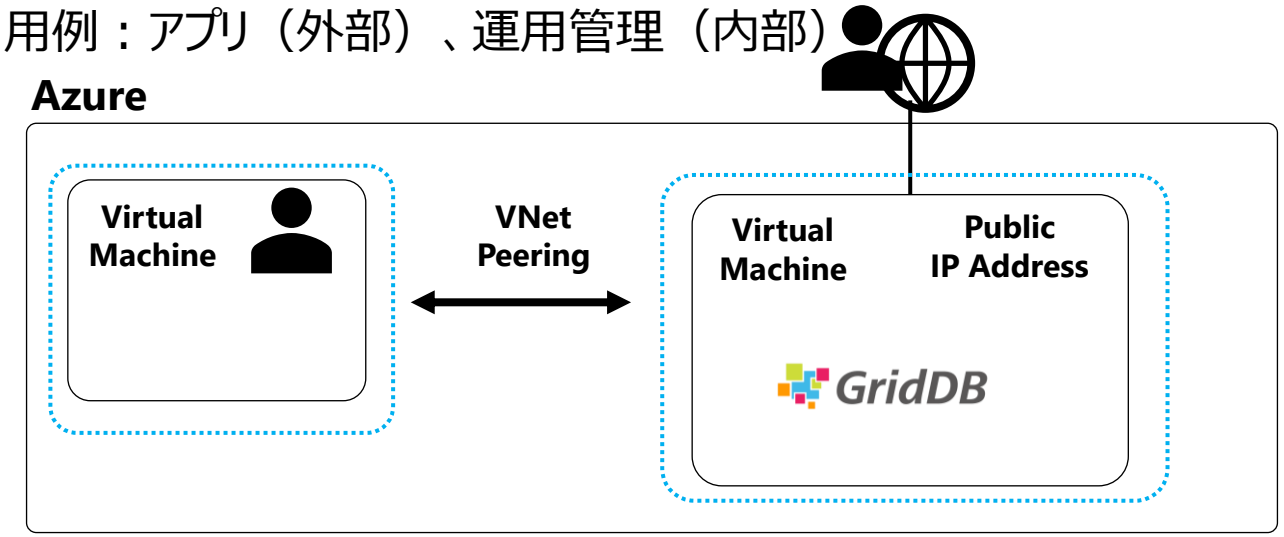
(B)内部接続
利用例：Webサーバとの連携



(C)外部接続
利用例：全て外部から操作

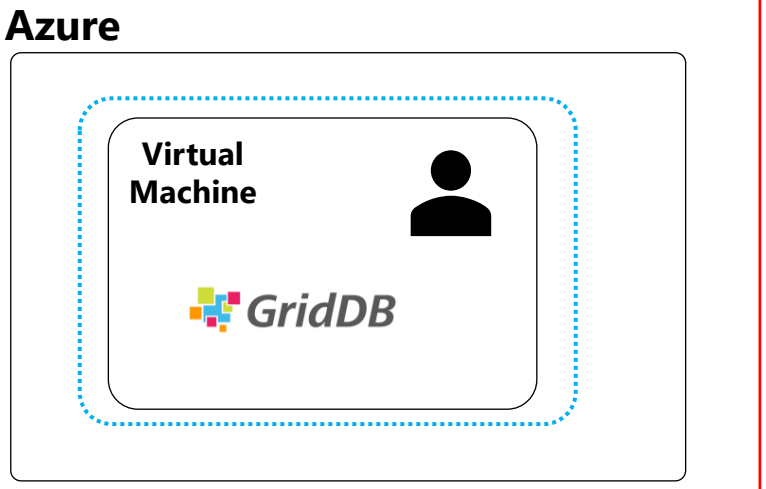


(D)外部・内部接続
利用例：アプリ（外部）、運用管理（内部）

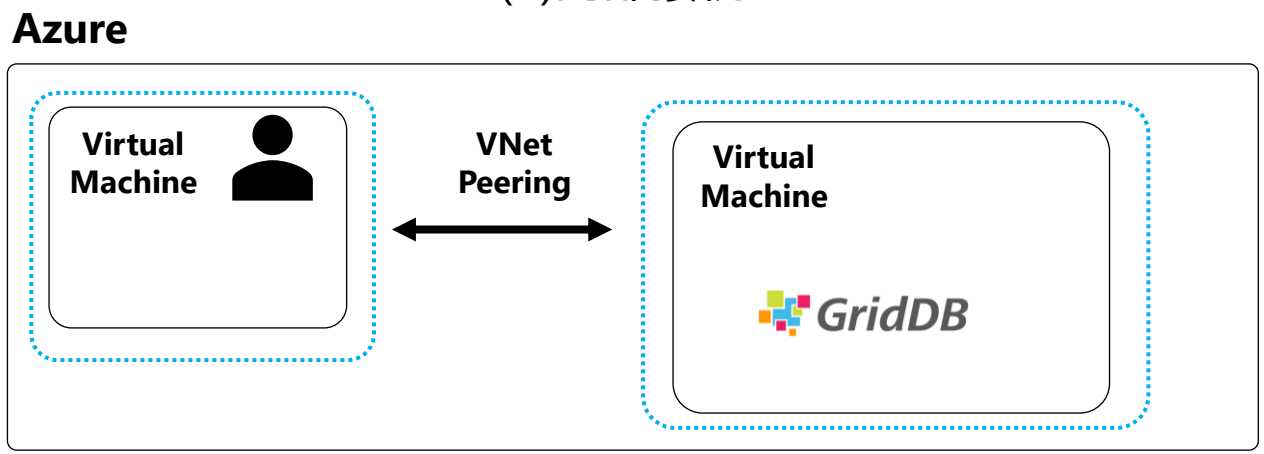


Azure上のGridDB利用形態

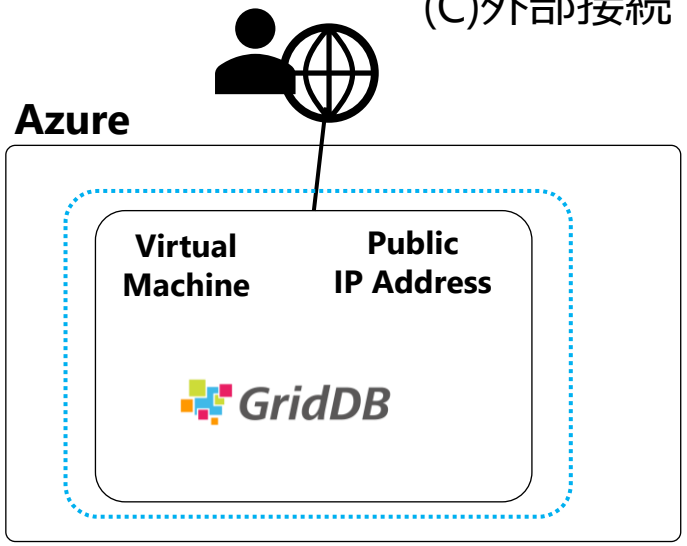
(A)ローカルアクセス



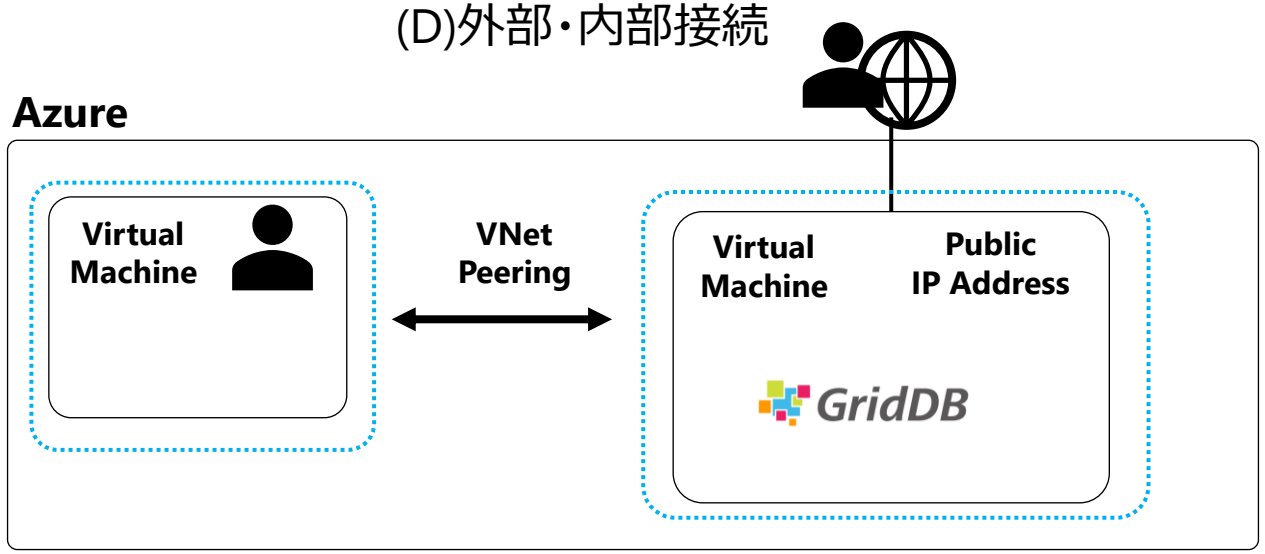
(B)内部接続



(C)外部接続



(D)外部・内部接続



GridDBのインストール＆起動の手順 (Ubuntuの例) ローカルマシン上

【インストール】

1. GridDBサーバのインストール

```
$ wget https://github.com/griddb/griddb/releases/download/v5.1.0/griddb_5.1.0_am64.deb
$ sudo dpkg -i griddb_5.1.0_amd64.deb
```

2. GridDB CLI（コマンドライン・インタフェース）のインストール

```
$ wget https://github.com/griddb/cli/releases/download/v5.0.0/griddb-ce-cli_5.0.0_am64.deb
$ sudo dpkg -i griddb-ce-cli_5.0.0_amd64.deb
```

【起動】

3. GridDBのサービス起動

```
$ sudo systemctl start gridstore
```

4. CLI起動

```
$ sudo su - gsadm
$ gs_sh
>
```

※GridDBサービスの停止

```
$ systemctl stop gridstore
```

GridDBのインストール＆起動の手順 (Ubuntuの例) ローカルマシン上

【インストール】

1. GridDBサーバのインストール

```
$ wget https://github.com/griddb/griddb/releases/download/v5.1.0/griddb_5.1.0_am64.deb
$ sudo dpkg -i griddb_5.1.0_amd64.deb
```

2. GridDB CLI（コマンドライン・インタフェース）のインストール

```
$ wget https://github.com/griddb/cli/releases/download/v5.0.0/griddb-ce-cli_5.0.0_am64.deb
$ sudo dpkg -i griddb-ce-cli_5.0.0_amd64.deb
```

【起動】

3. GridDBのサービス起動

```
$ sudo systemctl start gridstore
```

4. CLI起動

```
$ sudo su - gsadm
$ gs_sh
>
```

設定なし、５つのステップだけで
CLIによるSQLなどの操作が開始できる。

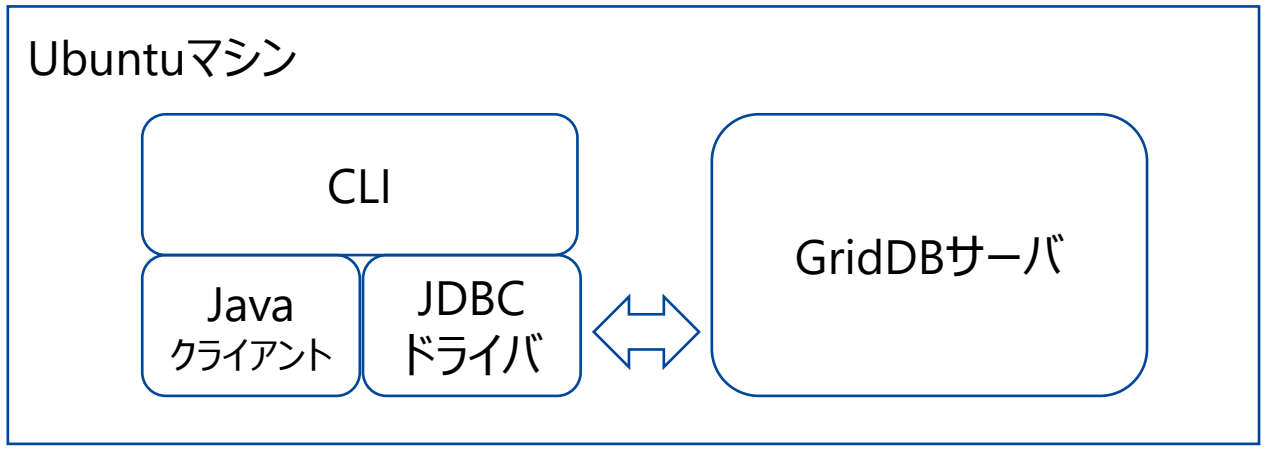
※GridDBサービスの停止

```
$ systemctl stop gridstore
```

＜動作環境の前提条件＞

- Azure上のVirtual Machine
- OSイメージはUbuntu 18.04。Javaインストール済

- 同一マシンに全ソフトウェアをインストール。ローカル実行
- GridDBのクラスタ名はmyCluster(デフォルト)
- GridDB管理者の名前はadmin、パスワードはadmin



※GridDBサーバ、Javaクライアント：https://github.com/griddb/griddb
※GridDB JDBCドライバ： https://github.com/griddb/jdbc
※GridDB CLI：https://github.com/griddb/cli

実行例１（SQL基本）

```
# テーブル作成
> create table t1 (c0 long, c1 long);

# データ登録
> insert into t1 values(1, 2);

# 検索
> select * from t1;
> get
```

※SQL文の先頭が下記文字列のいずれかである場合、コマンド名sqlを省略することができます。
select update insert replace delete create drop alter grant revoke pragma explain

実行例２（テーブルパーティショニング）：テーブル作成

装置

id	type	floor	room_no

```
CREATE TABLE equipTable (
  id INTEGER PRIMARY KEY, -- 装置ID
  type STRING,   -- 装置タイプ
  floor INTEGER, -- 設置階
  room_no INTEGER -- 設置ルームNo
);
```

センサデータ

date	id	value
インターバルハッシュパーティショニング： 分割幅30日、サブパーティション数6 パーティション解放：60日		

```
CREATE TABLE sensorTable (
  date TIMESTAMP, -- 日時
  id INTEGER, -- 装置ID
  value DOUBLE, -- センサ値
  PRIMARY KEY(date, id)
) WITH (
  expiration_type='PARTITION',
  expiration_time=60,
  expiration_time_unit='DAY'
) PARTITION BY RANGE (date) EVERY (30, DAY);
SUBPARTITION BY HASH(id) SUBPARTITIONS 6;
```

実行例２（テーブルパーティショニング）：データの登録

装置

id	type	floor	room_no
1	CAMERA	1	1
2	THERMO	1	1
｡｡｡			

センサデータ

date	id	value
2021-11-01T10:30:00Z	2	18.5
2021-11-01T10:30:00Z	3	20.0
｡｡｡		

```
INSERT INTO equipTable VALUES(1, 'CAMERA', 1, 1);
INSERT INTO equipTable VALUES(2, 'THERMO', 1, 1);
INSERT INTO equipTable VALUES(3, 'THERMO', 4, 3);
INSERT INTO equipTable VALUES(4, 'THERMO', 6, 2);
INSERT INTO equipTable VALUES(5, 'WATT', 1, 1);
INSERT INTO equipTable VALUES(6, 'WATT', 6, 1);
```

```
INSERT INTO sensorTable
  VALUES(TIMESTAMP('2021-11-01T10:30:00Z'), 2, 18.5);
INSERT INTO sensorTable
  VALUES(TIMESTAMP('2021-11-01T10:30:00Z'), 3, 20.0);
｡｡｡
```

JDBCドライバのインストールとサンプル実行

【インストール】
$ git clone https://github.com/griddb/jdbc
$ cd jdbc
$ ant

【サンプル実行】
 $ export CLASSPATH=${CLASSPATH}:./bin/gridstore-jdbc.jar
 $ cp sample/ja/jdbc/JDBCSelect.java .
←以降のスライドに記載のとおり、url部分を編集
 $ javac JDBCSelect.java
 $ java JDBCSelect

A.ローカルアクセス　（デフォルト）

クラスタ構成、APIからの接続：固定リスト方式

GridDBサーバ設定
クラスタ定義(gs_cluster.json)

```
"cluster": {
   "notificationMember":
   [{
            “cluster”:     {“address”: ”127.0.0.1", "port": 10010},
            “sync”:        {“address”: “ 127.0.0.1 ", "port": 10020},
            “system”:      {“address”: “ 127.0.0.1 ", "port": 10040},
            “transaction”: {“address”: “ 127.0.0.1 ", "port": 10001},
            “sql”:         {“address”: “ 127.0.0.1", "port": 20001}
```

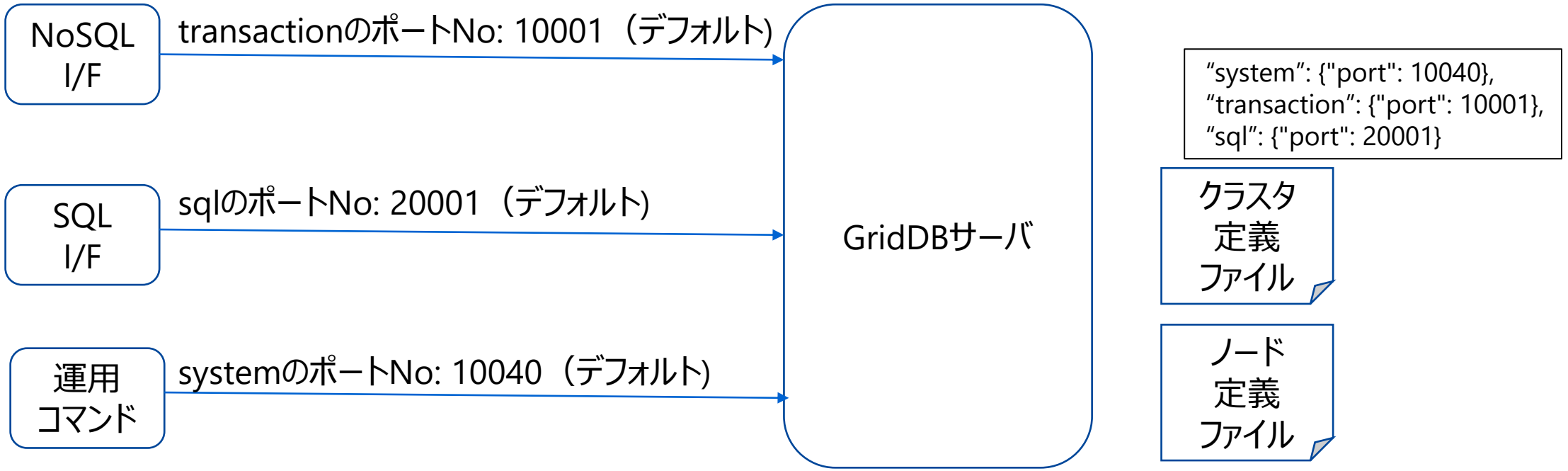
ノード定義(gs_node.json)

```
 “serviceAddress”: ”127.0.0.1",
```

APIからの接続

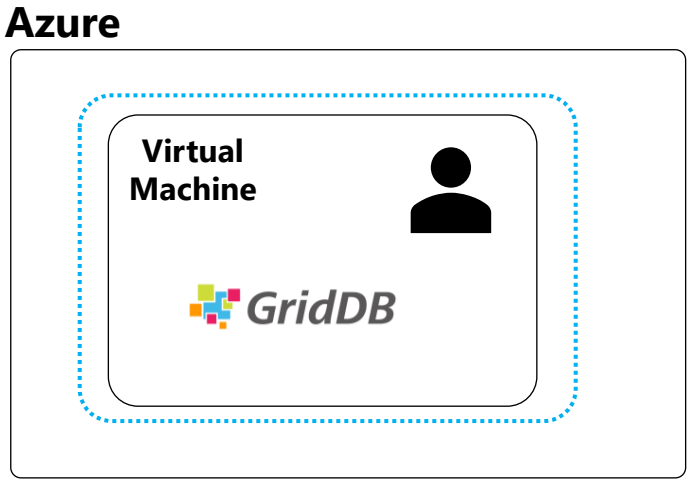
```
url = “jdbc:gs:///（クラスタ名）/?notificationMember=127.0.0.1:20001
```

GridDBサーバとの接続で利用されるポートNo

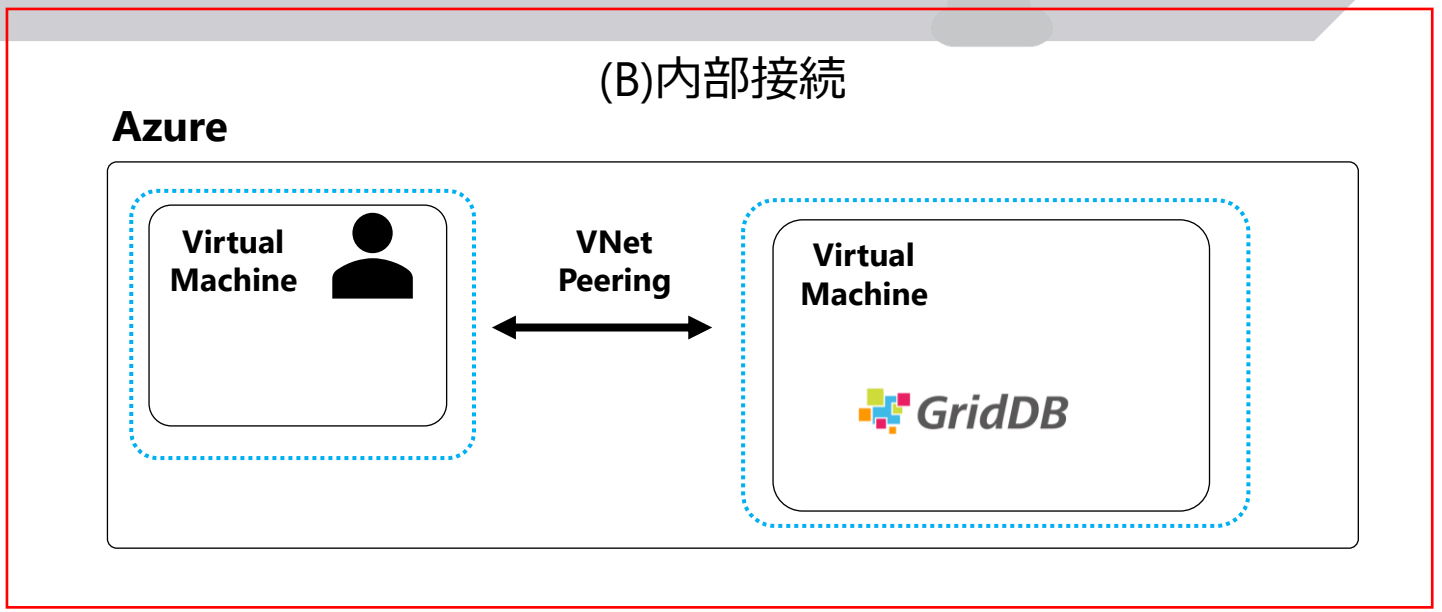


Azure上のGridDB利用形態

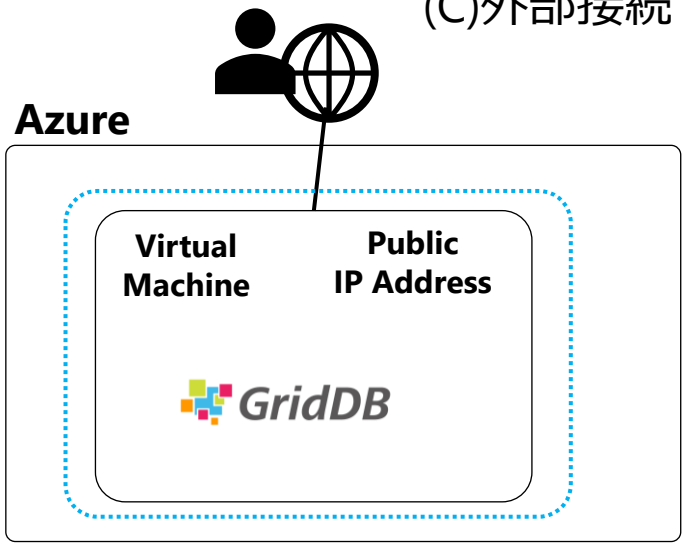
(A)ローカルアクセス



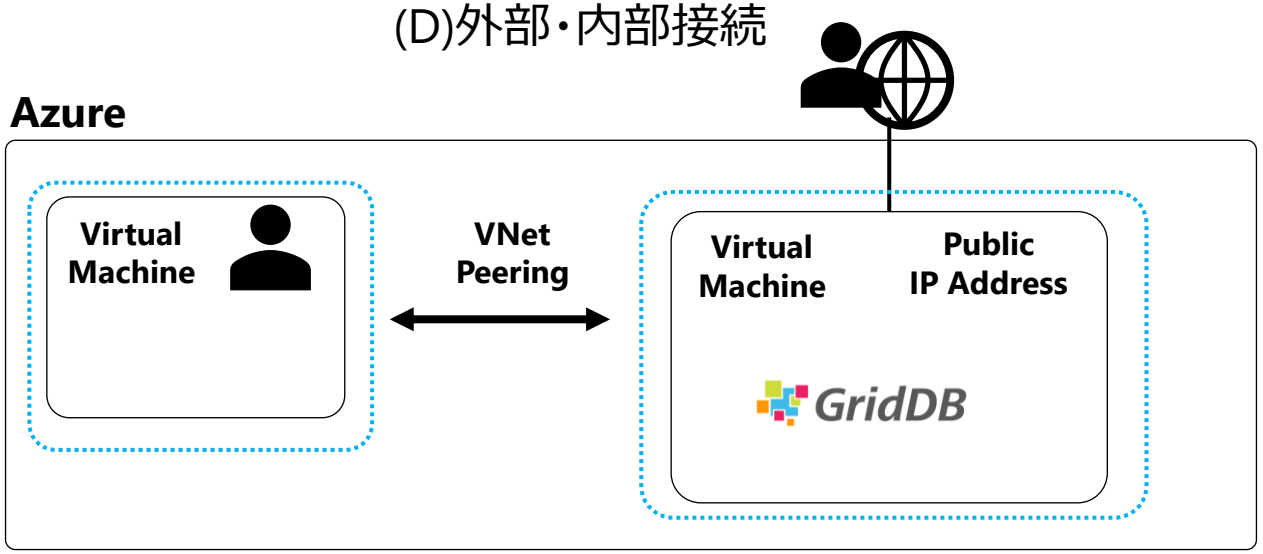
(B)内部接続



(C)外部接続



(D)外部・内部接続



リモートアクセス　（B.内部接続）

GridDBサーバ設定
クラスタ定義(gs_cluster.json)

```
"cluster": {
   "notificationMember":
   [{
            “cluster”:    {“address”: “（プライベートIP）", "port": 10010},
            “sync”:       {“address”: “（プライベートIP）", "port": 10020},
            “system”:     {“address”: “（プライベートIP）", "port": 10040},
            “transaction”: {“address”: “（プライベートIP）", "port": 10001},
            “sql”:        {“address”: “（プライベートIP）", "port": 20001}
```

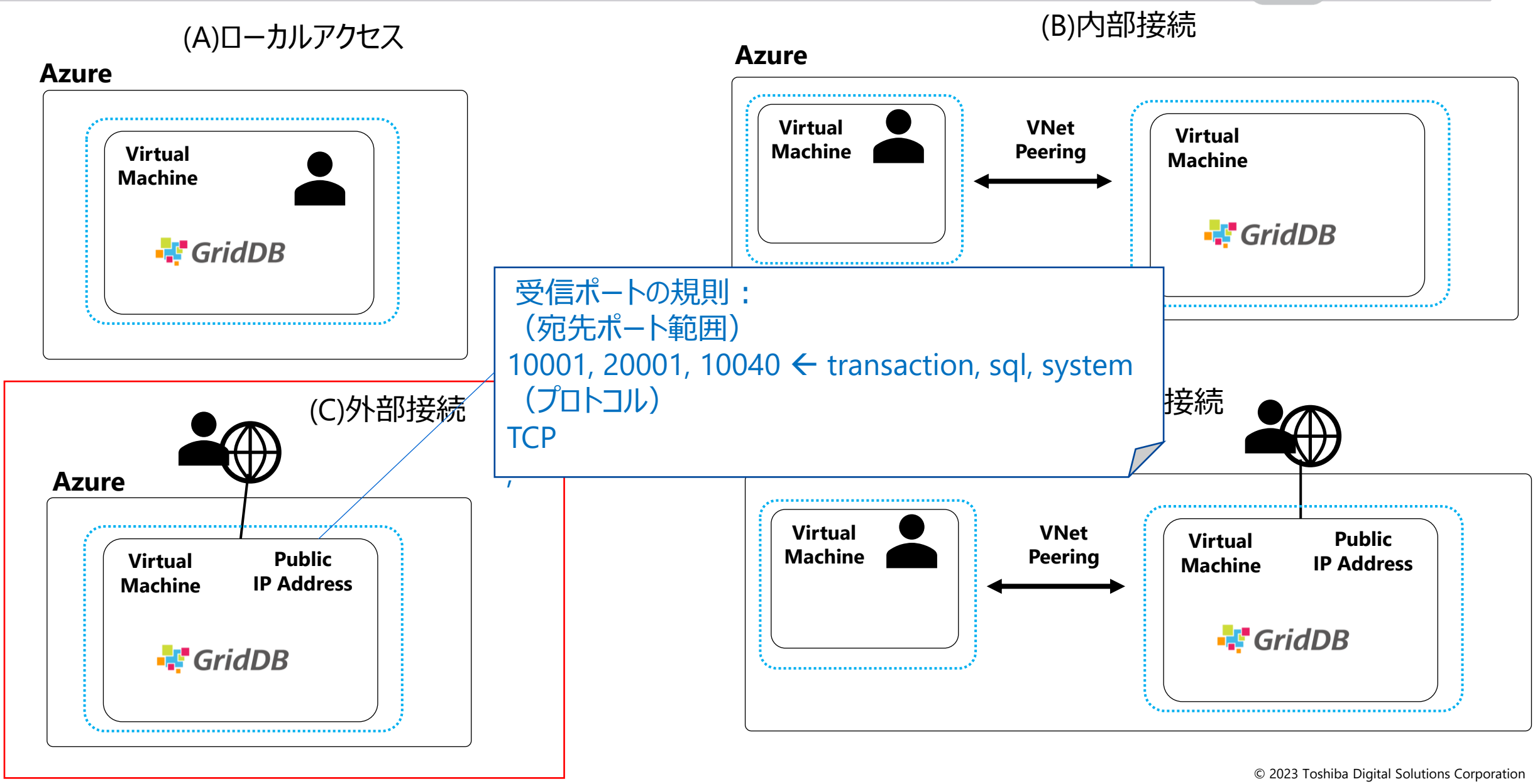
ノード定義(gs_node.json)

```
“serviceAddress”: “（プライベートIP）",
```

APIからの接続

```
url = “jdbc:gs:///（クラスタ名）/?notificationMember=（プライベートIP）:20001
```

Azure上のGridDB利用形態



リモートアクセス　（C.外部接続）

GridDBサーバ設定
クラスタ定義(gs_cluster.json)

```
"cluster": {
   "notificationMember":
   [{
            “cluster”:    {“address”: “（パブリックIP）", "port": 10010},
            “sync”:       {“address”: “（パブリックIP）", "port": 10020},
            “system”:     {“address”: “（パブリックIP）", "port": 10040},
            “transaction”: {“address”: “（パブリックIP）", "port": 10001},
            “sql”:        {“address”: “（パブリックIP）", "port": 20001}
```

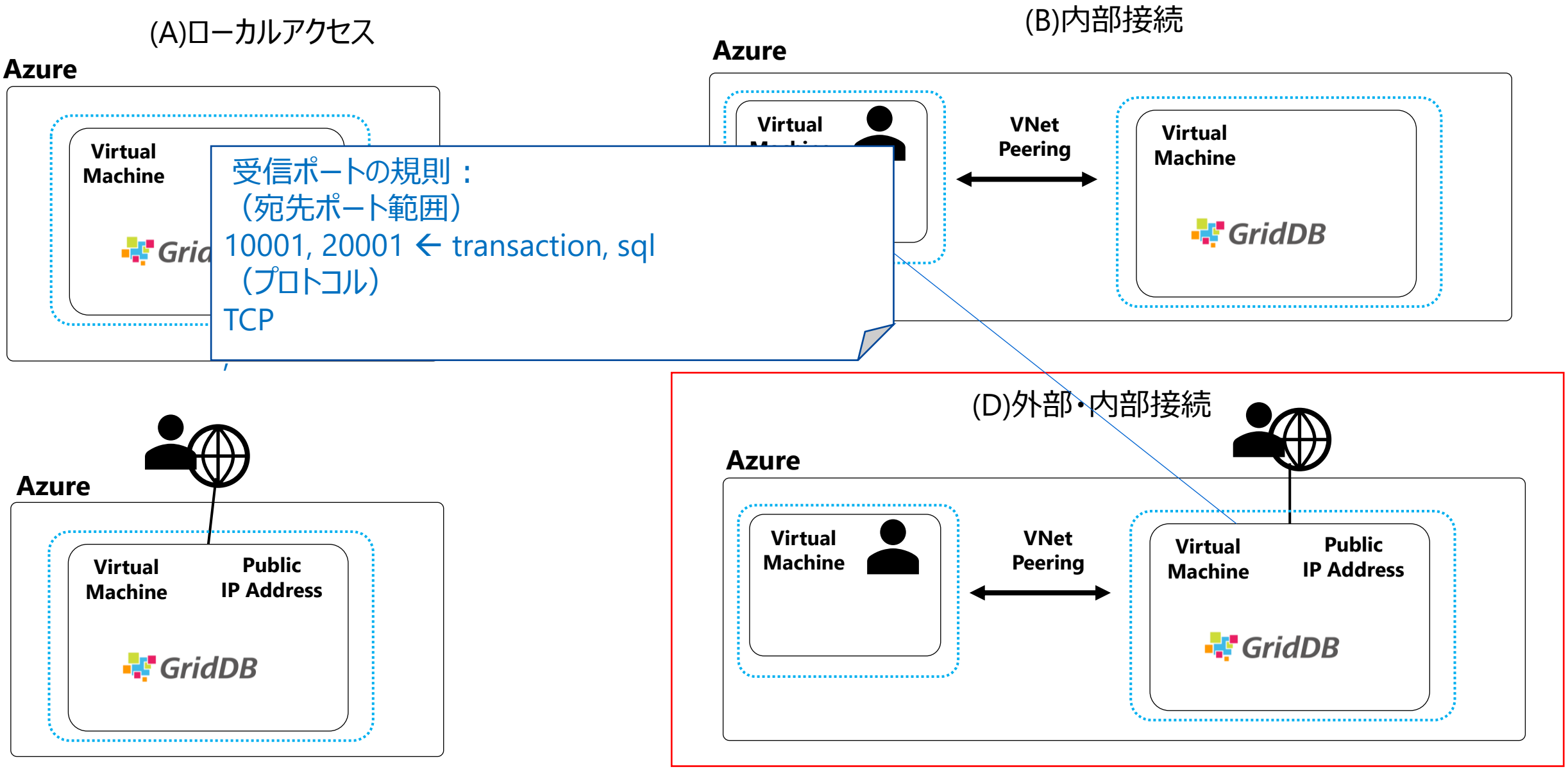
ノード定義(gs_node.json)

```
“serviceAddress”: “（パブリックIP）",
```

APIからの接続

```
url = “jdbc:gs:///（クラスタ名）/?notificationMember=（パブリックIP）:20001
```

Azure上のGridDB利用形態



リモートアクセス　（D.外部接続と内部接続）

GridDBサーバ設定
クラスタ定義(gs_cluster.json)

```
"cluster": {
   "notificationMember":
   [{
            “cluster”:    {“address”: “（プライベートIP）”, "port": 10010},
            “sync”:       {“address”: “（プライベートIP）”, "port": 10020},
            “system”:     {“address”: “（プライベートIP）”, "port": 10040},
            “transaction”: {“address”: “（プライベートIP）”, "port": 10001},
            “sql”:        {“address”: “（プライベートIP）”, "port": 20001}
           “transactionPublic”: {“address”: “（パブリックIP）”, "port": 10001},
            “sqlPublic”:       {“address”: “（パブリックIP）”, "port": 20001}
```

ノード定義(gs_node.json)

```
 “serviceAddress”: “（プライベートIP）”,
 "transaction":{
   “publicServiceAddress”:“（パブリックIP）”,...       },
"sql":{
   “publicServiceAddress”:“（パブリックIP）”,...     },
```

APIからの接続
・外部経由の場合

```
url = “jdbc:gs:///（クラスタ名）/?notificationMember=（パブリックIP）:20001“
&connectionRoute=PUBLIC"
```

・内部経由の場合

```
url = “jdbc:gs:///（クラスタ名）/?notificationMember=（プライベートIP）:20001
```

クラウドでの利用方法（まとめ）

	(A)ローカルアクセス	(B)内部接続	(C)外部接続	(D)外部/内部接続
GridDBサーバ設定： クラスタ定義 （gs_cluster.json）	そのまま	notificationMemberにプライベートIP指定	notificationMemberにパブリックIP指定	notificationMemberにプライベートIP指定 ＋ transactionPublicとsqlPublicにパブリックIP指定
GridDBサーバ設定： ノード定義 （gs_node.json）	そのまま	serviceAddressにプライベートIP指定	serviceAddressにパブリックIP指定	serviceAddressにプライベートIP指定 ＋ transactionとsqlのservicePublicAddressにパブリックIP指定
API(JDBCなど)からの接続	notificationMemberに127.0.0.1指定	notificationMemberにプライベートIP指定	notificationMemberにパブリックIP指定	notificationMemberに(内部)プライベートIP指定 (外部)パブリックIPとconnectionRoute=PUBLICの指定

03

（ご参考）GridDB Cloudのご紹介

GridDB Cloudとは

GridDB CloudはGridDBのクラウドサービスです

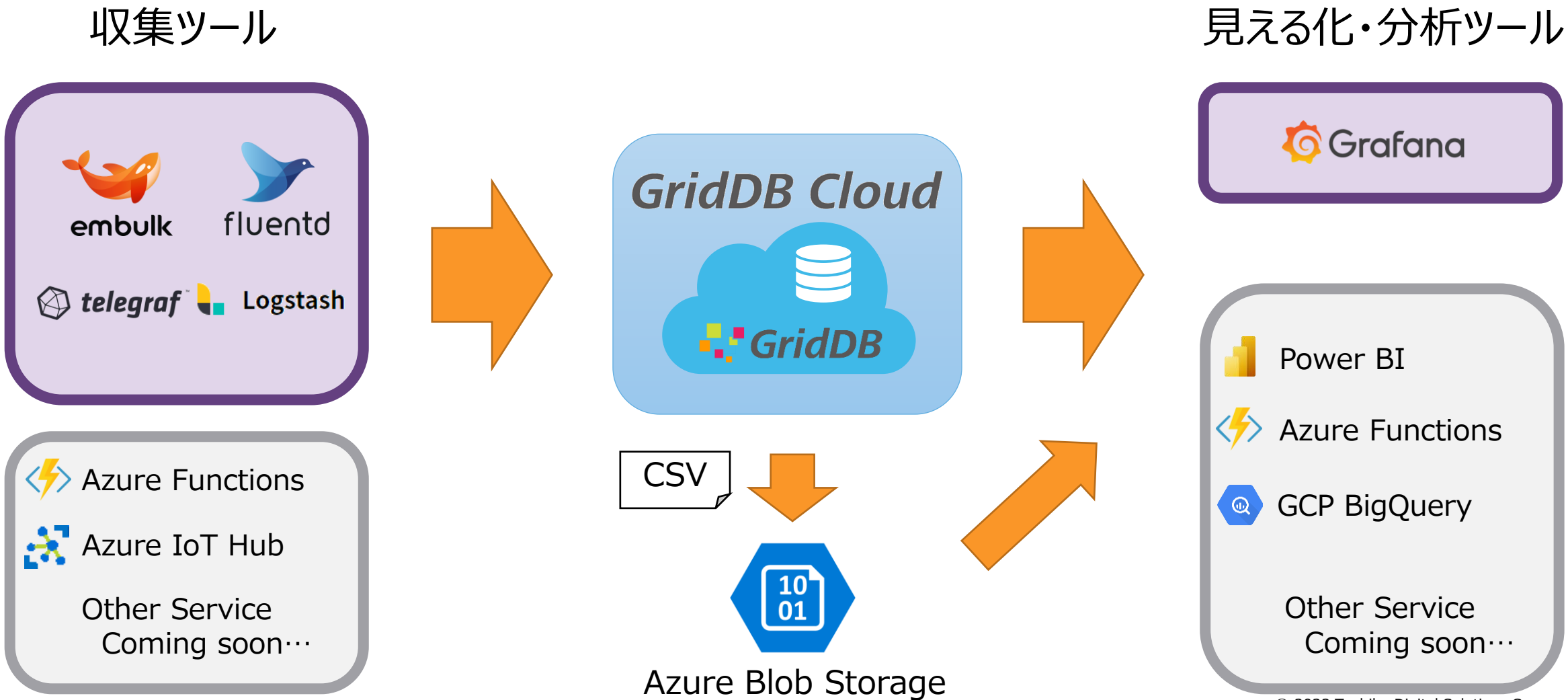
POINT 1 パブリッククラウドで稼働するマネージドサービス

POINT 2 クラウドネイティブアプリと簡単・高速に連携

POINT 3 データ収集やデータの見える化機能が充実

データ収集ツールや見える化ツールとの連携

単なるDBaaSではなく様々なツールと連携しクラウドデータ基盤を目指す



GridDB Cloudを無料で使ってみませんか？

griddb cloud | 検索

URL：https://form.ict-toshiba.jp/download_form_griddb_cloud/

GridDB Cloud関連のコンテンツ

- Logstash Output Pluginを使ってGridDBデータベースにSyslogメッセージを送信する
  - https://griddb.net/ja/blog/logstash-griddb/
- GridDBクラウドとPythonを用いた風力発電の分析
  - https://griddb.net/ja/blog/wind-turbine-analysis-using-griddb-and-python/
- 物価とインフレの高騰-GridDBクラウドとPythonによる分析
  - https://griddb.net/ja/blog/sky-rocketing-prices-inflation-an-analysis-using-griddb-and-python/
- 地球外生命体 – GridDBクラウドとPythonによる解析
  - https://griddb.net/ja/blog/alien-life-on-earth-analysis-using-griddb-and-python/
- GridDB クラウド Version 1.2 の紹介
  - https://griddb.net/ja/blog/using-griddb-cloud-version-1-2/
- GridDBクラウドの紹介
  - https://griddb.net/ja/blog/an-introduction-to-griddb-cloud-2/
- GridDB CloudでPower BIを使う
  - https://griddb.net/ja/blog/power-bi-with-griddb-cloud/
- 【入門】GridDB Cloud に VNetを使って触れてみよう！
  - https://qiita.com/gahoh/items/8dc8d81eec89a7c1bed1
- 【入門】GridDB Cloudにcurlを使ってWeb APIで触れてみよう！
  - https://qiita.com/gahoh/items/6c766e64c2c2c7aab81d
- 【入門】GridDB Cloud にPostmanを使ってWeb APIで触れてみよう！
  - https://qiita.com/gahoh/items/f45141ef56e90030d453

など

04

OSS活動

主なOSS活動

① GridDB本体の機能強化

② 主要OSSとの連携強化

③ APIの拡充

④ GitHub以外のサイトからの情報発信

- パッケージ
- デベロッパーズサイト（WP、ブログなど）
- SNS

⑤ 主要OSSリポジトリへのコントリビュート

⑥ プラットフォームの拡充

⑦ その他

- OSCなどカンファレンス参加
- ハンズオン無料セミナー

OSS活動の全体イメージ

性能測定 | 収集 | 分散処理 | 可視化 | Webアプリ | 分析 | AI/機械学習 …

④GitHub以外のサイトからの情報発信
PyPI/npm/Maven/Packagist/…

⑥プラットフォームの拡充
CentOS＋
Ubuntu、openSUSE
Windows、MacOS
Docker

②主要OSSとの連携強化

Spark コネクタ
Fluentd/Grafana/Redash プラグイン
YCSB コネクタ
Kafka コネクタ
Hadoop MapReduce コネクタ

WebAPI
Python/Node.JS/Go/PHP/Ruby/Perl/Rustクライアント

③APIの拡充

Javaクライアント
JDBCドライバ
Cクライアント

①GridDB本体の機能強化
GridDB V5.1 CE(Community Edition)

https://github.com/griddb/
GitHub

⑤主要OSSリポジトリへのコントリビュート

デベロッパーズサイト

https://griddb.net/

griddb net | 検索

- **アプリケーション開発者向けのサイト**
- **様々なコンテンツを公開**
  - ホワイトペーパ
  - ブログ

  など

最近のブログ

- Introducing the Rust Client for GridDB (2023/1)
  - ✓ 2022/10ソース公開したRust言語のクライアントライブラリを使ったブログ
  - ✓ https://griddb.net/en/blog/introducing-the-rust-client-for-griddb/
- Stream Data with GridDB and Kafka (2023/1)
  - ✓ 2022/9ソース公開したApache Kafkaコネクタを使ったブログ
  - ✓ https://griddb.net/en/blog/stream-data-with-griddb-and-kafka/

など

ツイッター

https://twitter.com/griddb_jp

twitter griddb	検索

- GridDBに関するリリース、イベント、などをお知らせします。
  (日本国内向け)

05

まとめ

まとめ

- GridDBはビッグデータ・IoT向けのデータベースです。
- GridDBの概要とクラウドでの使い方、オープンソース活動についてご紹介しました。
  - 今後も様々な拡張、拡充を進めて参ります。

GridDBのオープンソース版(GridDB CE)を是非とも使ってみてください。
https://github.com/griddb/

TOSHIBA

付録

各エディションの違い

- インタフェースはほぼ同じ
- クラスタ構成の有無の違い

項目	機能	Community Edition	Enterprise Edition	Cloud
サポート			✓	✓
プロフェッショナルサービス			✓	✓
データ管理	時系列コンテナ	✓	✓	✓
	コレクションコンテナ	✓	✓	✓
	索引	✓	✓	✓
	アフィニティ	✓	✓	✓
	テーブルパーティショニング	✓	✓	✓
クエリ言語	TQL	✓	✓	✓
	SQL	✓	✓	✓
NoSQLインタフェース	Java	✓	✓	✓
	C言語	✓	✓	✓
NewSQL(SQL) インタフェース	JDBC	✓	✓	✓
	ODBC		✓	✓
WebAPI		✓	✓	✓
時系列データ	時系列分析関数	✓	✓	✓
	期限付き解放機能	✓	✓	✓
クラスタリング	機能クラスタ構成		✓	✓
	分散データ管理		✓	✓
	レプリケーション		✓	✓
運用管理	ローリングアップグレード		✓	
	オンラインバックアップ		✓	✓
	エクスポート / インポート	✓	✓	✓
	運用管理GUI		✓	✓
	CLIツール	✓	✓	✓
セキュリティ	信暗号化 (TLS/SSL)		✓	✓
	認証機能 (LDAP)		✓	
オンプレミス環境	オンプレミス環境	✓	✓	
クラウドサービス	クラウドサービス			✓

ご参考：GridDBに関する情報

- **GridDB GitHubサイト**
  - https://github.com/griddb/griddb/
- **GridDB デベロッパーズサイト**
  - https://griddb.net/
- **Twitter: GridDB (日本)**
  - https://twitter.com/griddb_jp
- **Twitter: GridDB Community**
  - https://twitter.com/GridDBCommunity
- **Facebook: GridDB Community**
  - https://www.facebook.com/griddbcommunity/
- **Wiki**
  - https://ja.wikipedia.org/wiki/GridDB
- **GridDB お問い合わせ**
  - OSS版のプログラミング関連：Stackoverflow(https://ja.stackoverflow.com/search?q=griddb)もしくはGitHubサイトの各リポジトリのIssueをご利用ください
    プログラミング関連以外：contact@griddb.netもしくはcontact@griddb.orgをご利用ください

griddb github	検索

griddb net	検索

twitter griddb	検索

ご参考：

- SQL（テーブルパーティショニング）の例
  - ✓ https://github.com/knonomura/griddb-docker/blob/master/SQLSamples.md
  - ✓ https://github.com/knonomura/griddb-docker/blob/master/SQLSamples2.md

- NoSQLインタフェースでバッチ処理等を使いたい場合
  - ✓ https://github.com/griddb/griddb/tree/master/sample/guide/ja

  のSampleMultiPut.javaなどを参照願います。

- DockerでGridDBを使いたい場合
  - ✓ https://github.com/griddb/griddb-dockerのDockerfile
  - ✓ https://hub.docker.com/u/griddbのDockerイメージ

  を参照願います。