

衛星データ解析技術研究会 技術セミナー（応用編）

Webアプリケーションの開発技術の習得

第六回 2025/08/01

担当講師：田中聡至

アジェンダ

デプロイ編

ジオな体験を記録に残そう！（実践編&前回からの課題）

→ 先に少しだけアイデアワークショップをやります。

今回使うのは

AppRun β (さくらインターネット) : コンテナ環境

Xserver VPS : ベアメタル環境

(ハンズオンとしては講師に追従することを必要としませんが、もし、一緒の速度でやりたい場合はそれぞれのアカウントをご用意ください。)

(電話番号とクレジットカードの番号が必要となります。)

アプリケーションのデプロイ

どこで動かすか

どうやって動かすか

何を動かすか

どうやって動かすか / 何を動かすか

- 静的サイトホスティング
- 抽象化インフラ(動的サイトホスティング ... とは言わないかも)
- サーバレス運用
- VPS運用
- コンテナレジストリ

色々な場所でシステムを動かそう！

Linux環境下で動かす

Linux環境下にNode.jsなどが動く環境を用意して動かします。

そもそもLinux慣れてない方向けのアイデア

- Cloud Shell ([Google Cloud](#)、[さくらインターネット](#))を試す
- Dockerを利用して、Linuxに直接構築する

VagrantやVMwareを入れる

💡 **Docker**を利用して、コンテナ内のLinuxにログインできます。

`docker run -it --rm -v $(pwd):/app expresson` のような形でコンテナに入れます。(sshで接続します)

`docker run` コマンドにはさまざまなサブコマンドが用意されていて、ログイン時の設定や利用したい環境変数、ユーザ設定などを記載することもできる他、これらを Dockerfile内に記載することも可能です。

別タブでコンソールを開いて、

```
# コンテナから外部へファイルをコピー  
docker cp container_name:/path/to/file /host/destination
```

とすると、起動中のコンテナに対して状態をセーブすることができます。

本番環境: Docker Volume

開発環境: バインドマウント(/mnt)

一時的な保存: コピー(`docker cp`)コマンド

と覚えておくと便利です。(もともとコンテナとホストマシン(Win|Mac)で共有するフォルダを指定しておくことをお勧めします。)

AWSで動かす

EC2 は基本的にVPSのように使うことができます。

sshとして接続する際はAWSサービスの外側に対して接続可能なIPアドレスがListenしている必要がありますが、ガイドラインを見る限り現在でもそんなに接続に手間はないようです(.pemファイルを渡すだけ。)

(講師としてはちゃんとRoute53などを用意した方がいいのかと思っていましたがそこまで仰々しいパスを用意しなくても良い様子)

SSH クライアントを使用して Linux インスタンスに接続する - Amazon Elastic Compute Cloud

ちなみにWindowsマシンをAWSで動かして、リモートデスクトップを用いて、GUIで接続する方法もあります。

RDP を使用した Windows インスタンスへの接続 - Amazon Elastic Compute Cloud

同様の方針で、LinuxもGUIで接続したい場合は、ソフトウェアとしてGUIを導入することができます(仮想デスクトップというワードで検索すると良いです。**Lubuntu**などが軽くておすすめ。VNCで接続する方法もあります。)

💡 日本語環境だと多くの場合、Virtualを「仮想の」と訳すことが多いですが、原義としては「実質的な」という意味で捉えた方が良いでしょう。
そもそもGUI自体もプログラム(プラグイン)であるので、仮想デスクトップという単語自体に問題があるかもしれません。

最小構成



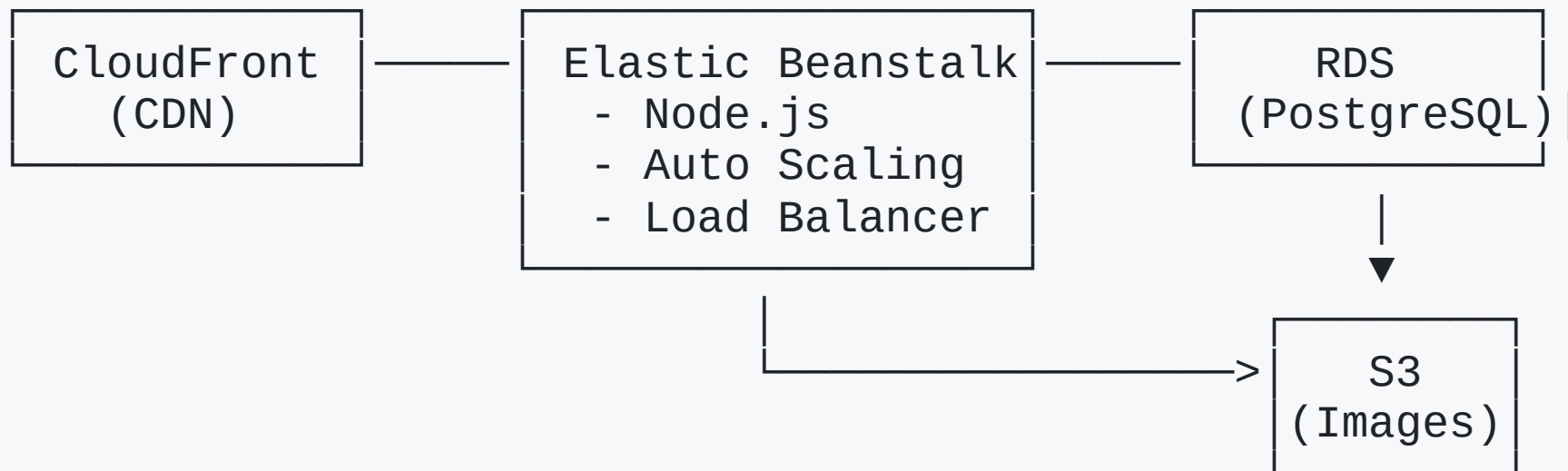
[AWS Elastic Beanstalk](#)を使うのが今風だそうです。

レンタルサーバのサーバーサイドを含めた版。

root権限がなく、限定的な動作ができるサーバーと捉えてください。

Heroku、Fly.io、Render、Railwayなど独立したPaaSが存在します。

Elastic Beanstalkを中心に考えるならこんな感じ？



OaaSとは？

Platform as a Service をはじめとするエンタープライズのサービス群を指します。

- Infrastructure as a Service
- Platform as a Service
- Backend as a Service

などなど

正確な切り分けは難しいですが、サーバーを置くためのサービスと考えると良いでしょう。

モバイルのバックエンドに特化して、MBaaS を謳うサービスもあります。

エッジで(サーバレスとして)動かす

サーバレスアーキテクチャとは？

→ 特定の機能のみを実行する組み合わせで構成されるアプリケーション
(○○Function のような名前のサービスであることが多いです。)

著名クラウドサービスの区分け

	VPS	非rootサーバー	サーバレス
Amazon Web Services	EC2	Beanstalk	Lambda
Google Cloud	Compute Engine	App Engine	Cloud Functions
Azure	Virtual Machines	App Service	Azure Functions

Linuxサーバーを作成し、講義中に構成したサービスを動かして見よう！

XServerを利用します。

[無料VPS | XServer VPS](#)

メールアドレス登録 → 電話番号認証 → サーバーイメージ作成 という手順を踏みます。

デモンストレーション

💡 任意のコマンドライン(CLI)を用意しましょう

sshに限って言えば、WindowsだとTeratermが便利で、また、VSCodeからリモートマシンに接続して操作、コーディングを行うことも可能です。
GitbushやPuTTYを利用してsshする場合もこれに準拠します。

[Visual Studio Code で SSH を使用してリモート マシン上で開発を行う - Training | Microsoft Learn](#)

GUIが存在しないのと、Webアプリケーションが動けば良いので、4GBで足りないことはほぼないと思います。

無料VPSのお申し込み

ご利用サービス名	XServer VPS (無料VPS)	
サーバー名	必須	<div>vps-2025-08-01-10-09-19</div> <p>【半角】英数字 50文字以下、記号は「-(ハイフン)」 「_(アンダーバー)」のみ使用可能</p>
プラン	必須	<div><div>2GB メモリ 2GB CPU 2Core NVMe SSD 30GB</div><div>4GB メモリ 4GB CPU 3Core NVMe SSD 30GB</div></div>

OS

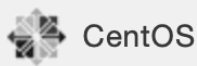
アプリケーション

イメージタイプ ②

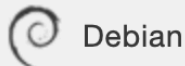
必須



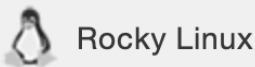
25.04 (64bit) ▼



Stream 10 (64bit) ▼



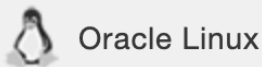
12 (64bit) ▼



10.0 (64bit) ▼



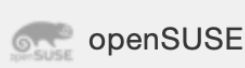
10.0 (64bit) ▼



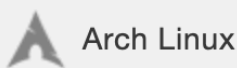
9.6 (64bit) ▼



42 (64bit) ▼



Leap 15.4 (64bit) ▼



2024.08.01 (64bit) ▼

rootパスワードは十分に強力なものを設定しましょう。(講師はターミナルなどでランダムな文字列を生成するコマンドを利用しています。ランダムと言いつつパターンがあったらハックされるのでは??)

rootパスワード 

必須

.....



半角英字、半角数字、半角記号 (\\^\$+~*/|000.,?!_=&@~%#;'") から2種類以上を含んだ9文字以上70文字以内

キーを作成する 

登録方法

☐ 自動生成

☒ インポート

SSH Key 

名前

ssh鍵の作り方

前ページの「SSH Key > インポート」で貼り付ける鍵の作り方を示します。

****公開鍵(.pubとつく方)****を貼り付けます。

```
$ ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (~/.ssh/id_ed25519): Xserver-20250801
Enter passphrase for "Xserver-20250801" (empty for no passphrase):
Enter same passphrase again:
```



無料VPSサービスは、不正利用防止のため、お申し込み時にクレジットカード情報のご登録をお願いしております。サービス利用中に、ご登録いただいたカードへ料金が請求されることはありませんので、ご安心ください。

カード番号 ②

1234 5678 9012 3456

有効期限

08

月 /

2025

年

セキュリティコード 

123

確認画面へ進む

[＜戻る](#)

申し込み完了後、**ダッシュボード**より、ブラウザでコンソール(CLI)を立ち上げることができます。

登録時に指定したパスワードは**root**ユーザーのもので、

Login : root とし、その後にパスワードを入力します。

上記の図では、一度ログインに失敗していますが、**welcome~~** が出るとログイン成功です。パスワードはコンソール上に表示されません。

Linux環境においては**root**ユーザーのみが用意されていることが多く、この環境でも同様で、まずは**一般ユーザー**を作る必要があります。

```
# ユーザー一覧を確認
ls /home
# 新規ユーザー作成
sudo adduser username
# ユーザーのパスワード設定
sudo passwd username
# sudoグループに追加（Ubuntu/Debian系）
sudo usermod -aG sudo username
```

デモでは **tnk** ユーザーを作成しました。

デモではssh用の公開鍵を事前にサーバに渡していましたが、これはサーバを建てた際に `/root/.ssh` に配置されている `authorized_keys` ファイルに存在していますが、一般ユーザの領域においては存在しないので、

```
# rootで実行
cp /root/.ssh/authorized_keys /home/user1/.ssh/
chown user1:user1 /home/user1/.ssh/authorized_keys
chmod 600 /home/user1/.ssh/authorized_keys
chmod 700 /home/user1/.ssh
```

のように、`user1` ユーザーとしてログインする際にも公開鍵を利用するようにします。

さらに、X-ServerのWebコンソールより、sshで利用する22番ポートが利用できる状態になっているかを確認めます。

その後、たとえば `root` アカウントだと

```
ssh -p 22 -i ~/.ssh/Xserver-20250801 root@x162-43-31-95.static.xvps.ne.jp
```

のような形でログインができます。

```
ssh -p {ポート番号} -i {どの秘密鍵を提示するか} {ユーザ名}@{ホスト名やIPアドレス}
```

のような構成です。

デモ中のハマりどころとしては、

```
Are you sure you want to continue connecting (yes/no/ [fingerprint])?
```

と表示した際に空白のままEnter押してしまいましたが、ここで `yes` を明示的に入力すべきでした。

ここで、接続が確立されるとローカル側の `~.ssh/known_hosts` に通信が記載されます。(VScodeなどで確認が可能です。)

Windowsでは通常は `c:\Users\ユーザー名\.ssh\known_hosts` に記載されます。

特定のポートが接続可能かどうかはローカルマシンから

```
curl -v telnet://リモートサーバーアドレス:ポート番号
```

のような形式で確かめるか、telnetコマンドなどが現在でも利用できます。
また、リモートサーバーにWebコンソールからログインしている場合には、

Netid	State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port	Process
udp	UNCONN	0	0	127.0.0.54:53	0.0.0.0:*	
udp	UNCONN	0	0	127.0.0.53%lo:53	0.0.0.0:*	
tcp	LISTEN	0	4096	127.0.0.54:53	0.0.0.0:*	
tcp	LISTEN	0	4096	127.0.0.53%lo:53	0.0.0.0:*	
tcp	LISTEN	0	4096	0.0.0.0:22	0.0.0.0:*	👉
tcp	LISTEN	0	4096	:::22	:::*	👉

のように確かめることができます。

sshでアクセスができることが確かめられたら、ローカルで動いているアプリケーションをリモートサーバーにアップロードします。

scpやsftpソフトウェアによるファイルの送信はssh通信を使うので、前述の処理が必要でした。

以下、scpコマンドで、第5回までに完成したソフトウェアをアップロードします。

```
scp -r -i ~/.ssh/Xserver-20250801 ~/2025-RemoteSensingSeminar/2025-07-25/handson/3-map user1@x162-43-31-95.static.xvps.ne.jp:/home/user1
```

↑鍵の指定 ↑ローカル ↑リモート

scpコマンドが成功したら、sshを行って、サーバーで作業をします。
今回のアプリケーションは、node.jsとnpmが動けば良いので、
一般ユーザでログインした状態で

```
sudo apt install nodejs  
# node -v で確かめる  
sudo apt install npm  
# npm -v で確かめる
```

をした後に、プロジェクトフォルダに移動 (`cd 3-map`) して、

```
npm install
```

すると実行準備が整います。

💡 node.js環境なので、Javascriptのライブラリ集がダウンロードすればこれくらいの環境構築で済みますが、Pythonや、Rubyなどを利用したい場合は、`apt` などを用いてこれらの実行環境を整える必要があります。

実行

事前に22番ポートが利用できる設定をしましたが、この時点で、同様にWebコンソールから、3000ポートや443ポートにアクセスできるように設定しておきましょう。

```
npm run dev
```

ポート3000での待受が始まったら、実際にアクセスできるか確かめましょう！

```
# 講義中にアクセスした例 : 現在は無効化されています。  
http://x162-43-31-95.static.xvps.ne.jp:3000/
```

デモで対応しなかった範囲

- rootユーザーによるSSHは禁止
- NGINXの導入、SSL対応

特に、SSL対応を行わないと、GeoLocationAPIなどの情報取得が制限される他、DBへのPOSTのアクセスが拒否されることも多いです。

(ここでLocalhostだと `http://` だったはず...と思った方は鋭くて、Localhostは挙動が特殊です。また、localhostでもSSL通信を利用する方法もあります。)

一定額で使いやすいVPS

- ConoHa VPS
- さくらのVPS
- AWS Lightsail

Docker編

AppRun β コントロールパネル
コンテナレジストリー一覧

アプリケーションをDocker用に修正

<https://github.com/alt9800/2025-RemoteSensingSeminar/tree/main/2025-08-01/handson/expressonDocker>

具体的にはDockerfileやDocker-compose.ymlなどを修正しています。

AppRun β版 | さくらのクラウド マニュアル

コンテナレジストリ | さくらのクラウド マニュアル

気をつけるべきこと

Dockerコンテナのサイズ

シングルバイナリにビルドできるような実行環境(Goなど)が相性がよく、一方でJVMのようなインタープリタを実行する環境が少し相性が良くないと感じます。

ビルドイメージ

arm CPUを積んだPCが増えてきているので注意しておく、コンテナを登録する(アップロードする)レジストリがx86_64アーキテクチャである場合、そのままarm64やM1 Macなどでビルドしただけでは動作しません。

そのため、Dockerコンテナをビルドする際に明示的にx86_64を指定してあげましょう。

環境変数

コンテナ環境に限りませんが、マシン側で環境変数を平文で設定したくはないので、ビルドの際にコマンドとして与えてあげると、DockerFileなどに記載するよりもよりセキュアだと思います。

基本的にはDockerは「**PCごとの実行環境の差分を吸収してくれる**」ための仮想化環境である、と説明していましたが、
実際にはホストマシン側のリソースを利用して計算を行っている都合上、CPUアーキテクチャが違くと命令セットが違いため、それぞれのCPUに最適化されたコンテナでは利用するカーネルで混乱してしまい実行がなされません。

アプリケーション層



Docker コンテナ



Docker Engine



ホストOS のカーネル ← ここでCPU命令を実行



CPU (ARM or x86_64)

まずは事前にコンテナレジストリにコンテナイメージを登録しましょう！

さくらインターネットにログイン後、

<https://secure.sakura.ad.jp/cloud/iaas/#!/appliance/containerregistry/>

にアクセスし、

[左側メニュー]グローバル> コンテナレジストリの画面を開き、「+ボタン」から、コンテナイメージを追加します。

慣れないうちは権限を Pull & Push にしておきましょう。

石狩第1ゾーン

リソースを検索...

オプションヘルプ

サーバ

ストレージ

ネットワーク

アプライアンス

グローバル

エンハンスドLB

GSLB

DNS

コンテナレジストリ

シンプル監視

シンプルMQ

シンプル通知

EventBus

モニタリングスイート

セキュアモバイル

ショートメッセージ

コンテナレジストリ追加

キャンセル作成

名前

名称未設定

任意, 1~64文字

コンテナレジストリ名

example.sakuracr.jp

小文字の英字、数字、ハイフン(-)が利用できます。英字で開始し、英字または数字で終わる必要があります

独自ドメイン

myname.example.com

公開設定

☐ Push & Pull

☐ Pullのみ

☒ 非公開

認証なしでイメージにアクセスできるように設定できます

説明

任意, 1~512文字

タグ

任意, ENTERキーで追加

アイコン

- ▼

2025/08/01

日本語English

サウンドオフ

FEEDBACK

コンテナレジストリが作られたら、イメージをPushするためにユーザを作成しましょう。



コンテナレジストリ » 2025-08-01-maphandson

⚠️ コンテナレジストリを削除

📄 情報

👤 ユーザ

コンテナレジストリにアクセスできるユーザを最大1000件まで登録できます

<input type="checkbox"/>	#	ユーザ名	ユーザ権限設定	
<input type="checkbox"/>	1	tnk	All	 

✖ ユーザを削除

+ 追加

<https://github.com/alt9800/2025-RemoteSensingSeminar/tree/main/2025-08-01/handson/expressonDocker>

にも記載していますが、AppRunで動くコンテナはレジストリに登録しておく必要があり、講義のコード群だと以下のようなコマンドを用いてコンテナをビルドし、AppRunのレジストリにコンテナをアップロードします。

```
## さくらのコンテナレジストリの場合
docker login your-registry.sakura.ne.jp

## スクリプトに実行権限を付与
chmod +x build-and-push.sh

## 環境変数を設定してビルド
export REGISTRY_URL="your-registry.sakura.ne.jp"
export TAG="v1.0.0"
./build-and-push.sh
```

レジストリ名はレジストリページに記載のある、ユーザ自身が決めたURI(URL)です。

ライブコーディング中に出会ったエラー

```
=> ERROR [builder 4/6] RUN npm ci --only=production 2.2s
-----
> [builder 4/6] RUN npm ci --only=production:
1.159 npm warn config only Use `--omit=dev` to omit dev dependencies from the install.
2.032 npm error code EUSAGE
2.032 npm error
2.032 npm error The `npm ci` command can only install with an existing package-lock.json or
2.032 npm error npm-shrinkwrap.json with lockfileVersion >= 1. Run an install with npm@5 or
2.032 npm error later to generate a package-lock.json file, then try again.
2.032 npm error
2.032 npm error Clean install a project
2.032 npm error
2.032 npm error Usage:
2.032 npm error npm ci
2.032 npm error
2.032 npm error Options:
2.032 npm error [--install-strategy <hoisted|nested|shallow|linked>] [--legacy-bundling]
```

→ `npm ci --only=production` を `npm install` に変更

もしくは、`npm install` 実行しておくとCIに引っかかりません。

App Run側の設定

次は、レジストリに登録したコンテナをAppRun側で利用するように設定していきます。

<https://secure.sakura.ad.jp/cloud/#/catalog>

ホーム

プロジェクト

ユーザ

ユーザグループ

サービスプリンシパル

公開鍵

SSOプロファイル

イベントログ

ホストメンテナンス

さくらのクラウド

契約リソース

最適化サジェスト

利用料金管理

APIキー

請求情報

サービス間接続申し込み

さくらの専用サーバ PHY

さくらのクラウドホーム

「さくらのクラウドって何から始めればいいのか？」という方におすすめ！
[【始めよう！さくらのクラウド（1）】](#) 準備編：使い始めるときの操作の流れ のコラムを読む



さくらのクラウド

サーバやストレージなどの多彩なサービスが1時間単位から使える、高性能で低価格なIaaS型クラウドサービスです。



さくらの専用サーバ PHY

物理専有型のベアメタル型クラウドサービスです。申込から最短10分で利用可能。ハイスペック・大容量のサーバを専有でき、安定した性能を実現します。



オブジェクトストレージ

大容量のデータや大量のデータをスケラブルに保存できる、Amazon S3互換APIを備えた分散型クラウドストレージサービスです。



ウェブアクセラレータ

突発的なアクセスへの備えから日々の負荷軽減まで幅広く気軽にご利用いただけるCDNサービスです。負荷を分散させて表示を安定化できます。



AppRun β

インフラ管理を気にせず、コンテナレジストリからアプリケーションを簡単にデプロイできるサービスです。

コンテナのアクセス待受ポート番号は80のままでも、コンテナ前段のWebサーバがWell-knownポートを探してくれるようで、3000ポートをNode.jsが利用する設定でも問題なくアクセスできます。

AppRun β

アプリケーション作成 | [マニュアル](#) | | | |

アプリケーション

コンテナレジストリ

β版における注意事項と制限事項

フィードバック

アプリケーションの作成

コンテナレジストリに登録したイメージを使って、アプリケーションをAppRun β にデプロイできます。
事前に[さくらインターネットが提供するコンテナレジストリ](#) でユーザを作成の上、AppRun β にデプロイしたいコンテナイメージを登録しておきましょう。

アプリケーションの設定をしてください

アプリケーション名

アプリケーションを識別・管理するための名前

ポート設定

ポート

コンテナがアクセスを待ち受けるポート

自動スケーリング

アプリケーション全体の最大スケール数

最小 1、最大 10

先ほどのレジストリで決めたイメージのURLとイメージタグを指定します。
以下が例です。

```
clematis2025-08-01.sakuracr.jp/field-survey-app:latest
```

コンテナの構成情報を設定してください

コンテナ

変数

?

コンテナイメージ

example.sakuracr.jp/hello:latest

コンテナレジストリサービスに登録されているコンテナイメージ

?

コンテナレジストリ
アクセス設定（任意）

ユーザー名

コンテナレジストリのユーザー名

パスワード

コンテナレジストリのパスワード

- コンテナレジストリの公開設定を「非公開」にしている場合、「ユーザー名」「パスワード」を入力してください。

コンテナレジストリが非公開の場合、ここでユーザの情報を指定してあげるとセキュアにレジストリを扱えます。

コンテナを作成して、ステータスが「● 正常」になれば、
提示されるURL `https://app-{ハッシュ値}.ingress.apprun.sakura.ne.jp/` より、サーバーにアクセスできます。

お疲れ様でした。

ちなみに、ビルドしたDockerイメージは仮想マシン内に保存されていますが、

```
# Macの例  
~/Library/Containers/com.docker.docker/Data/vms/0/data/Docker.raw
```

必要がなくなればイメージおよび、ビルドキャッシュは適宜削除しましょう。

```
docker system df
```

で占有リソースを確認後、

```
docker system prune  
docker rmi field-survey-app:latest
```

などで適切にリソースを解放してあげましょう。

トレンドの話

Cloudflareを使おう！

CloudflareはCDNのみならず、無料範囲でサービス運営を始めやすく、AWS互換のKVS(あるいはRDBMS)やストレージサーバを持っており、ホスティングやサーバレス設計ができる基盤です。

(日本語のドキュメントがAWSに輪をかけて不足している印象はあります。)

SSL通信も無料範囲で提供されます。

実体をR2(オブジェクトストレージ)に置くよりも、Cloudflare Pagesを利用すべきですし、

さらにはWorkersを使うように推奨されています。

現在ではExpressがCloudflareでは動かないのですが、issueが立っているのでそのうち動くようになる可能性が高いです。(需要もありますし)

ExpressがCloudflare Workersで動くようになるかも

<https://github.com/cloudflare/workerd/pull/4549>

Cloudflare環境だと[Hono.js](#)が認証周りが非常に楽なのでこちらを利用すると良いと思います。

Supabaseを使おう！

FirebaseのOSS互換として語られることが多いですが、セルフホスト以外にも
<https://supabase.com> で無料枠でサービス運営を始めることができます。

用意されているDBがPostgreSQLを用いているので、高速なインデックスも利用できます。

Edge Functionsのサーバレス基盤も人気です。

(サービスの仕様上、デフォルトの設定で、クライアントサイドから操作できる範囲が広いので、セキュリティを考慮する必要があります。)

アイデア例

PLATEAUのデータを使っている...

データのコンバータが公開されており、glTFに変換するとWebベースの3Dオブジェクトとして取り回しがしやすいです。

一定範囲のみ必要であればBlenderで矩形で切り取りましょう。



マインクラフトに変換する

<https://github.com/Project-PLATEAU/plateau2minecraft>

- DEMを表示
- 天気予報図を作ろう
- **地図記号**を使って地形図を作ろう(地図帳でよく見るやつ)
- 空想地図コンバータ(OpenGeoFictionを使った空想地図作成など)

一番大きいニュース

AlphaEarth Foundations helps map our planet in unprecedented detail - Google
DeepMind

OSMからベクターマップを作ろう！

```
# osm2pgsqlでPostGISにインポート
osm2pgsql -c -d osm -S openstreetmap-carto.style japan.osm.pbf

# OpenMapTilesでタイル生成
docker run -v $(pwd):/data openmaptiles/openmaptiles-tools generate-vectortiles
```

```
# pg_tileservを使用（リアルタイム配信）
docker run -p 7800:7800 \
  -e DATABASE_URL=postgres://user:pass@host/db \
  pramsey/pg_tileserv
```

```
# ogr2ogrでGeoJSON経由
ogr2ogr -f GeoJSON output.geojson \
  PG:"host=localhost dbname=mydb" \
  -sql "SELECT * FROM buildings"
```

画像ファイルやshapefileを元にpbfを作成したい

→ tippecanoe

```
# 直接tippecanoeで処理
ogr2ogr -f GeoJSON - input.shp | \
tippecanoe -o output.pmtiles \
  --no-feature-limit \
  --no-tile-size-limit
```

この講義を通しての振り返り

昨年度のセミナー内容は、

Web開発初心者でもWebGISに触れるようにしよう！

というミッションの元に進めており、

今年度(2025年度)セミナーでは、そこから一歩進んで、**DBを含んだ永続的なデータの保存環境、Webへのデプロイを含めたWeb開発の全体像を学ぶ**ことを目指しました。

おまけ

今回の講義で紹介した書籍

Software Design総集編【2018～2023】 | 技術評論社

Amazon.co.jp: FOSS4G HANDBOOK（フォスフォーgerハンドブック） : OSGeo財団, 森亮（OSGeo財団日本代表） : 本

Pythonによる地理空間データ分析 - O'Reilly Japan

書籍としてお持ちした、PLATEAUアカデミーで配布される マニュアルは、G空間センサーやAIGIDにお尋ねするといただけるそうです。(2024年末の情報)

今年度2025年度も各地で開講されます。

PLATEAUアカデミー | (一社)社会基盤情報流通推進協議会

土地計画基礎調査で市町村と各測量会社さんが作られたデータに対して、各種属性や3Dモデルの情報の付加、CityGMLの検修などが対象範囲です。

非常に人手がかかる作業で、関わられる会社さんが増えることを応援しています。

オリジンとは？

通常、<https://example.com:443> と <https://example.com:8080> はそれぞれ別のオリジンと見なされ、あるオリジンに存在するリソースを、悪意あるスクリプトが別のオリジンから読み取ろうとすることを防ぐ仕組みがブラウザには存在しています。

JAMStackという考え方

講義でお伝えしたフロントエンドに関わる概念は、概ねJAMStackに則っています。
ある種のアーキテクチャ、アプローチと言えるでしょう。

JAMStackのJAMは JavaScript/APIs/Markup から成り立ちます。

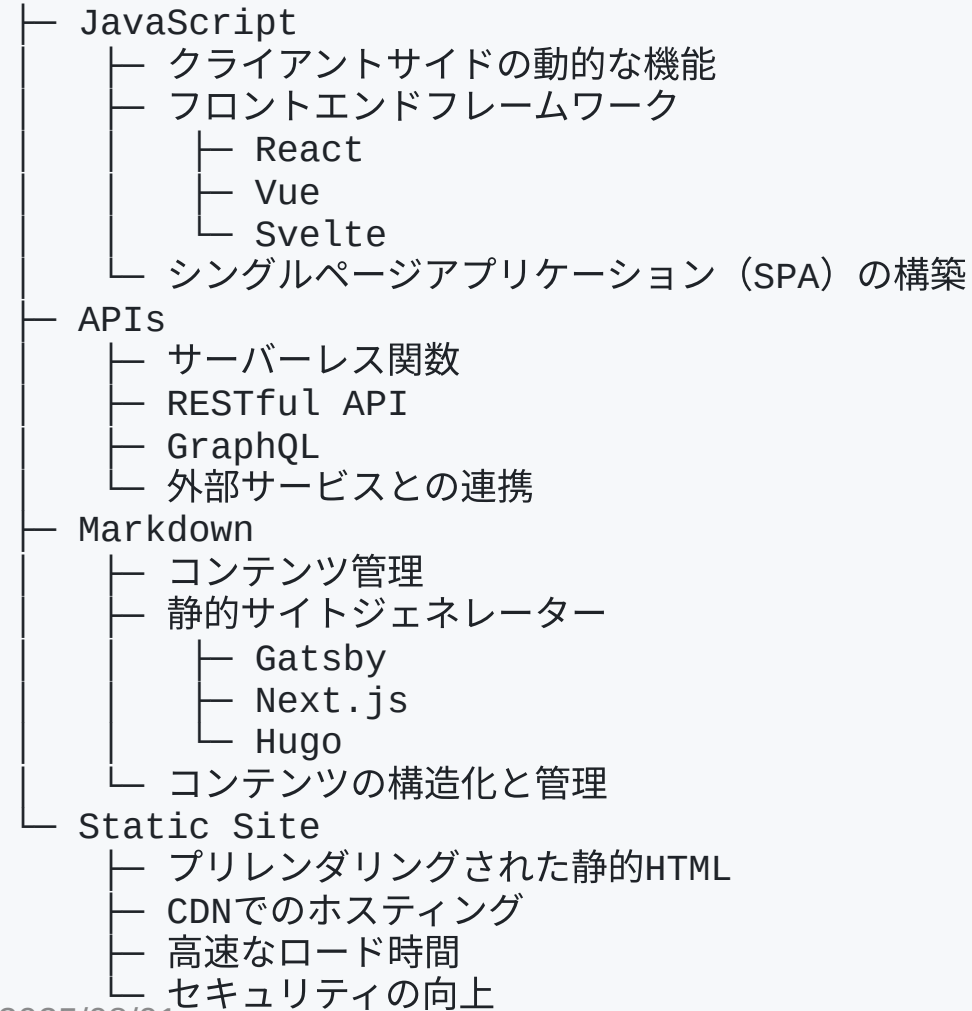
[Jamstackって何なの？何がいいの？ #JavaScript - Qiita](#)

以下のような便利ツールも推進団体が作っています。

[Static Site Generators - Top Open Source SSGs | Jamstack](#)

各要素をまとめるとこのような感じになるでしょうか？

JAMSTACKアーキテクチャ



動的なフロントエンドとは何か、とメモしておく、以下のような要素が当てはまるでしょう。

インタラクティブな要素

フォームのリアルタイムバリデーション

ドラッグ&ドロップ機能

モーダルウィンドウの表示/非表示

データの動的な更新

無限スクロール

リアルタイム検索結果の表示

ページ遷移なしでのコンテンツ更新

ユーザーインターフェース

タブの切り替え

アコーディオンメニュー

スライダー/カルーセル

状態管理

ショッピングカートの更新

ユーザー認証状態の管理

フィルターや並び替え機能

リアルタイム機能

チャットアプリケーション

通知システム

リアルタイム共同編集

アニメーション

スムーズなトランジション

スクロールエフェクト

インタラクティブな要素のアニメーション

確認メールの送信のおすすめ

認証において二段階認証で本人確認をするお話で、メール認証について触れましたが、

メールを送るためのAPIが公開されているサービスもあります。

ReSend / SendGrid

ポート転送の補足

最近のトレンドだとVPNも含めたトンネル網として[Tailscale](#)の利用が流行っているようです。

コンテナの補足

DockerDesktopの代替としては、OrbStack、Limaが人気があります。
AWSがFinchをかつて作っていたようですが...