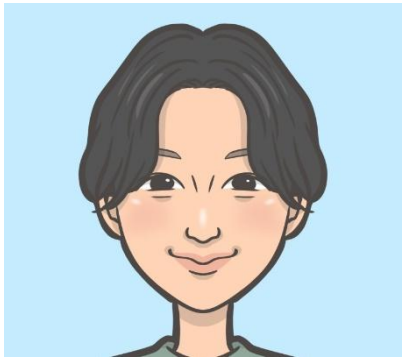
# イチから体験！
# NGINX ハンズオントレーニング
## ～認証編～

東京エレクトロン デバイス株式会社

# 自己紹介

# 自己紹介



**名前：**

**西川　常（ニシカワ　ジョウ）**

**経歴：**

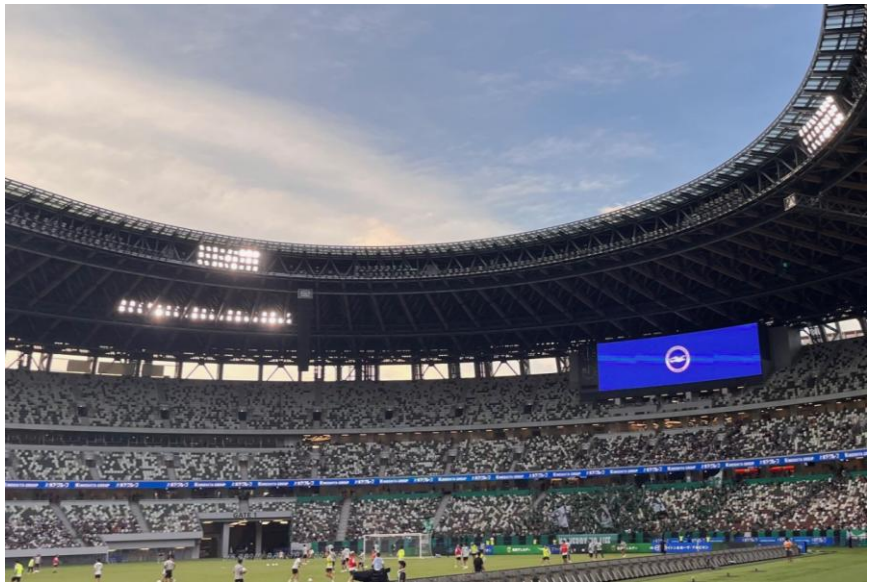
**ネットワークエンジニアとして設計・検証・構築**

**（L3SW大好き）**

**プリセールスエンジニアとして、F5製品を担当**

**趣味：**

**サッカー見てます　フロサポ**



# 東京エレクトロンデバイスについて

- F5の日本法人が出来る前からの一次代理店
- F5国内販売額複数年連続No.1の一次代理店
- 幅広い取り扱いラインナップ
  - F5 BIG-IP
  - F5 NGINX
  - F5 Distributed Cloud Services
- 保守契約ユーザーに対する手厚い付加価値サービス
  - 会員制Webサポートサイトの提供（製品FAQ、各種ドキュメントなど）
  - F5製品の重要情報をPush型でメール配信

  （脆弱性、基地の重大不具合及び改修情報、リリース情報など）

お気軽にご相談ください（お問合せフォームより）

https://cn.teldevice.co.jp/product/f5-nginx/

# Agenda

- ハンズオン環境の準備
- NGINX Plusについて
- mTLS証明書認証
- Basic認証
- JWTによる通信制御
- OIDCによる通信制御
- まとめ

# ハンズオン環境の準備

# ハンズオン環境 (UDF)

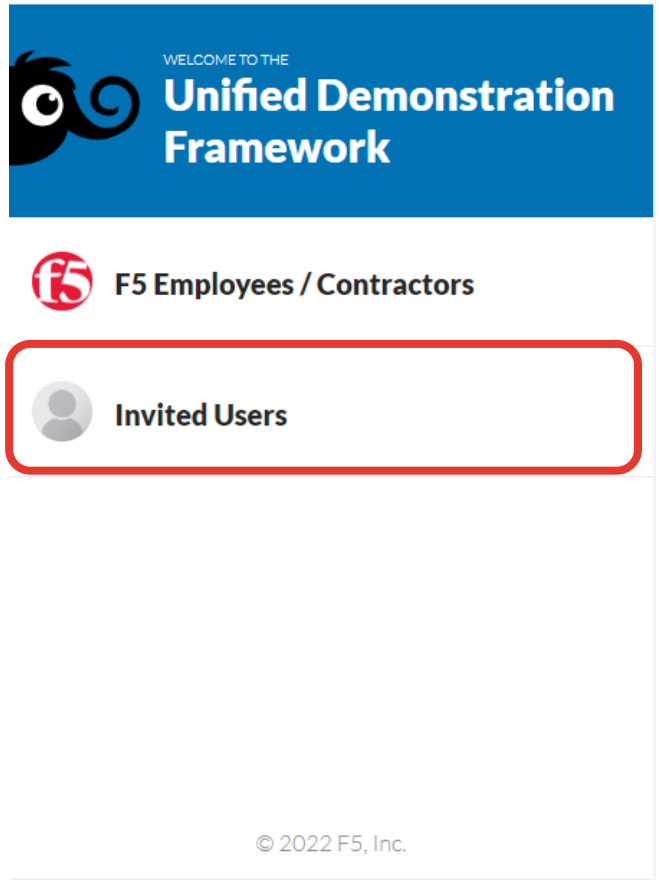
https://udf.f5.com

## ハンズオンガイド

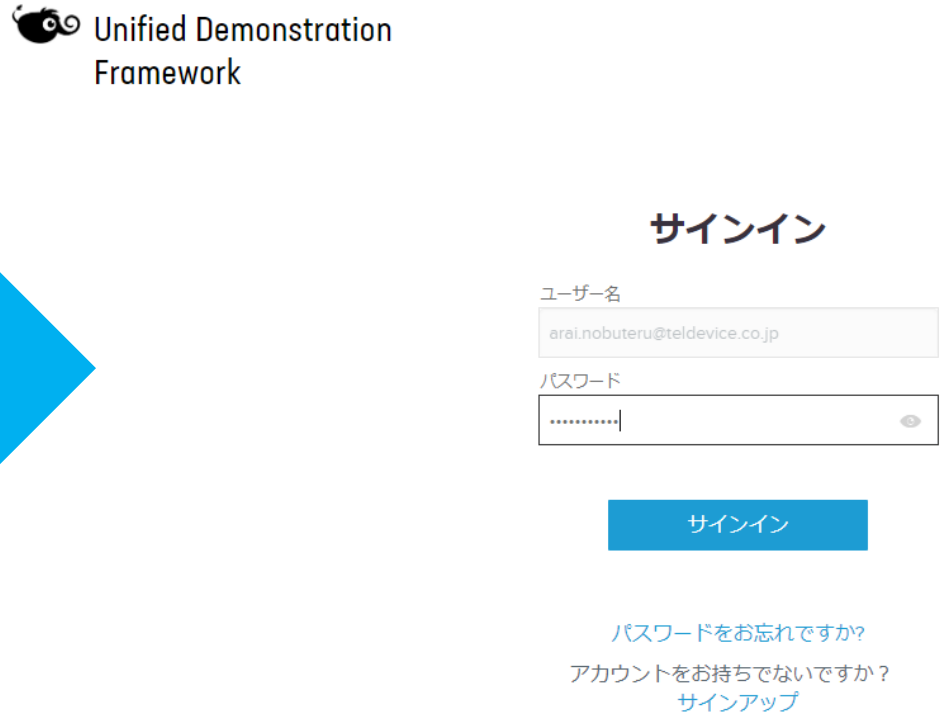
https://f5j-nginx-plus-lab2.readthedocs.io/en/latest/class1/module03/module03.html

# ハンズオン環境へのログイン

## “Invited Users” を選択します




## 登録したユーザーでログインします

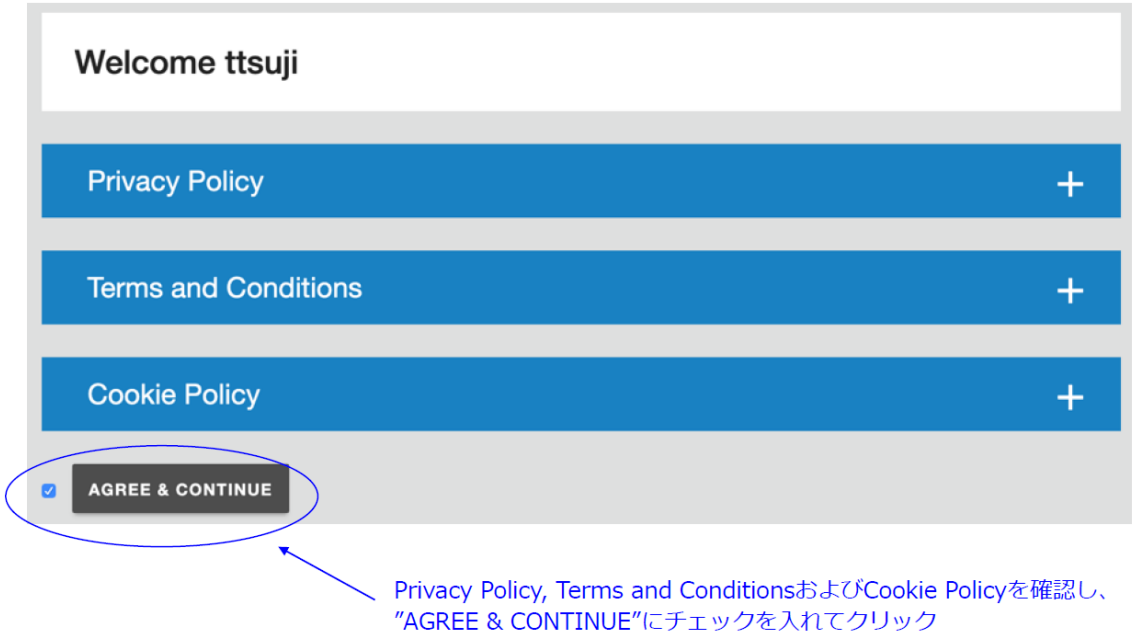


# ハンズオン環境へのログイン

## 2段階認証 します


## ポリシーに Agree します

※ 事前に実施されている場合は表示されません



# ハンズオン環境の準備

## トレーニング・セッションへの参加

招待されているトレーニング・セッションを選択して、
”LAUNCH”をクリック

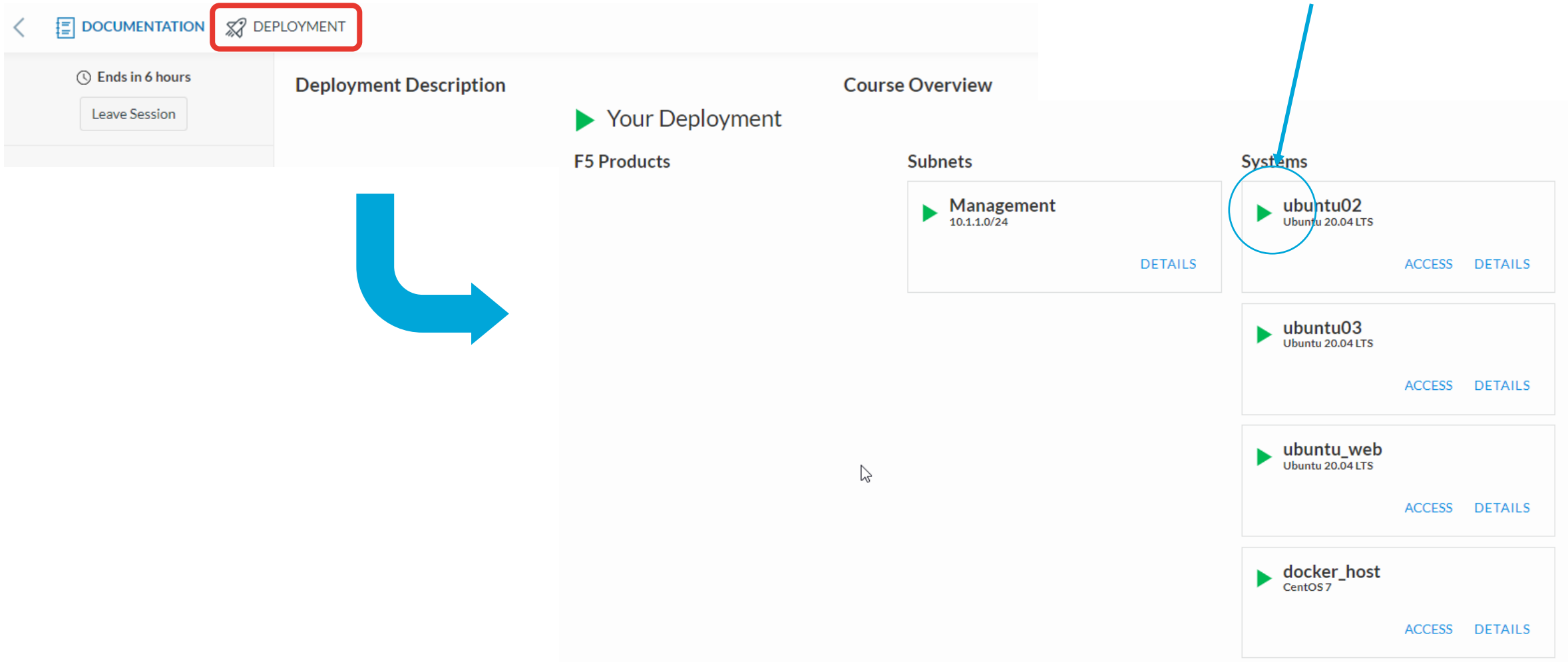


“Join”をクリック

# ハンズオン環境の準備

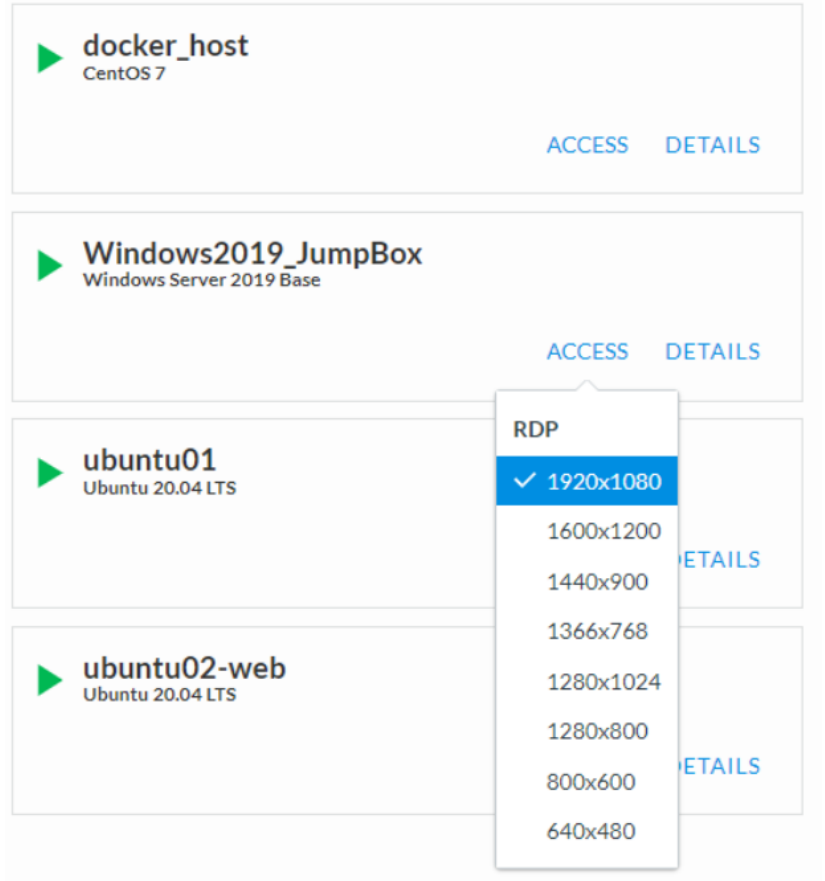
## DEPLOYMENT タブを選択します

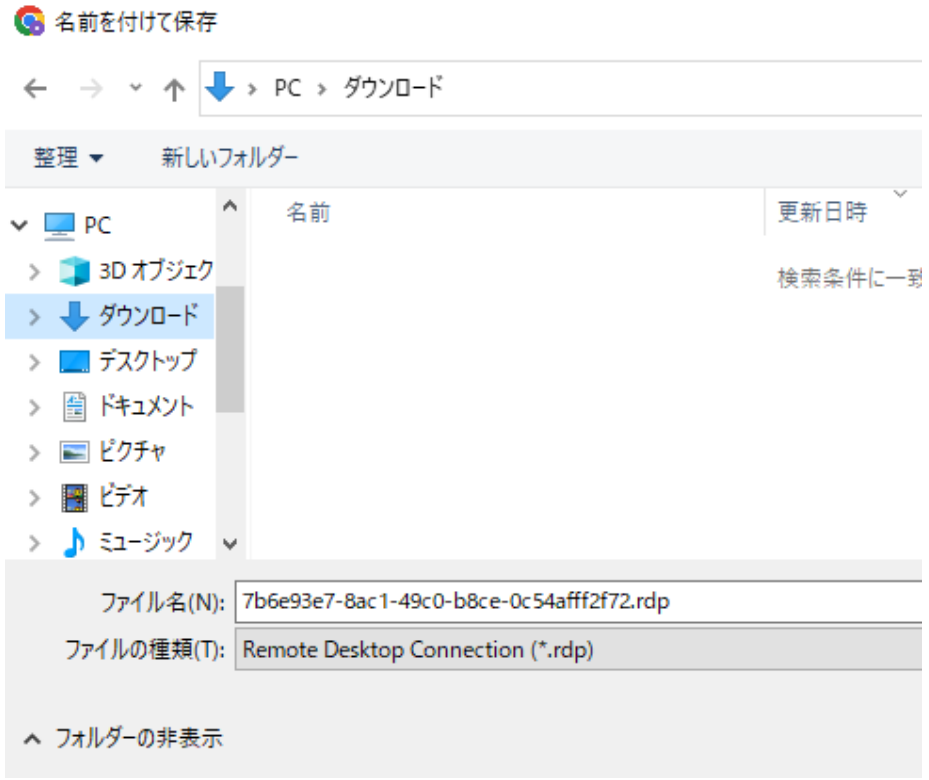
すべての system が 緑△ になれば準備完了です



# リモートデスクトップ接続

Windows2019_JumpBox の Access を選択して
RDP の画面サイズを選択して、RDP ファイルをダウンロードします



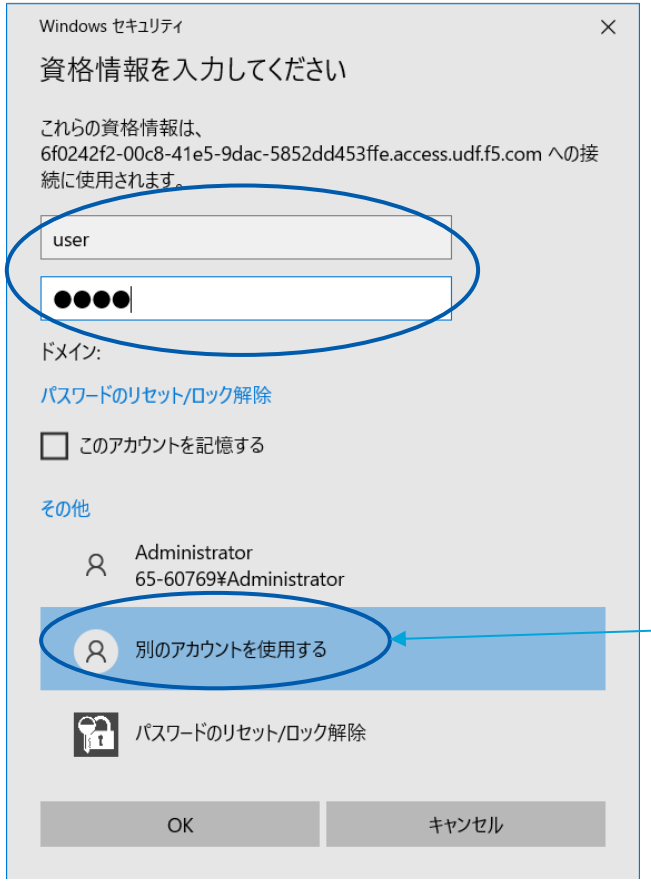



# リモートデスクトップ接続

RDP ファイルを実行して windows2019_jumpbox にログインします

User： **user**

Password：**user**

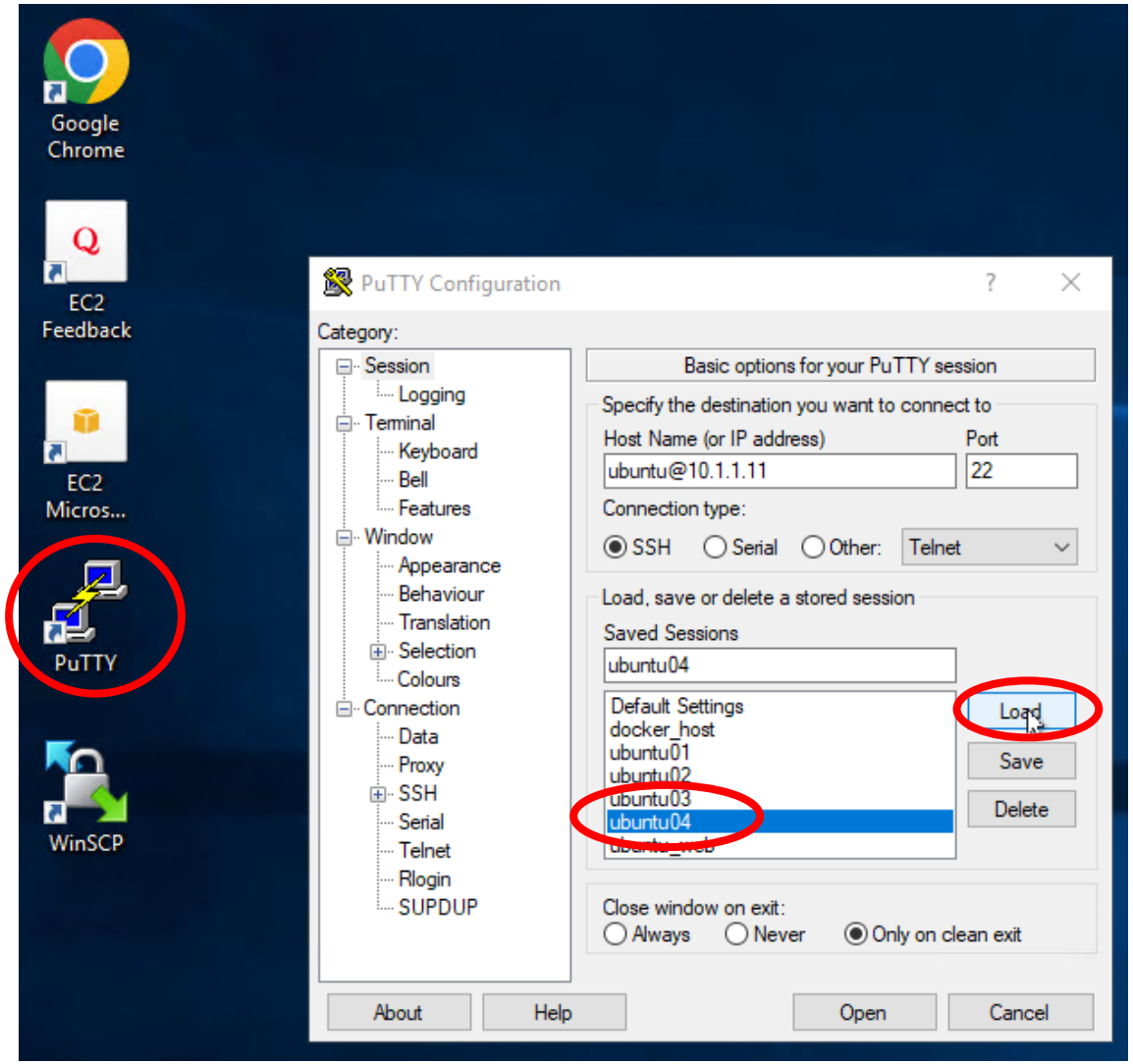



デフォルトは Administrator アカウントになっているため、
こちらを選択します。

# ubuntu04 ログイン

Putty を開いて **ubuntu04** にログインします

1) Putty アイコンを選択します
2) ubuntu04 を選択します
3) Load ボタンを選択します
4) Open ボタンを選択すると
   ubuntu04にログインできます。



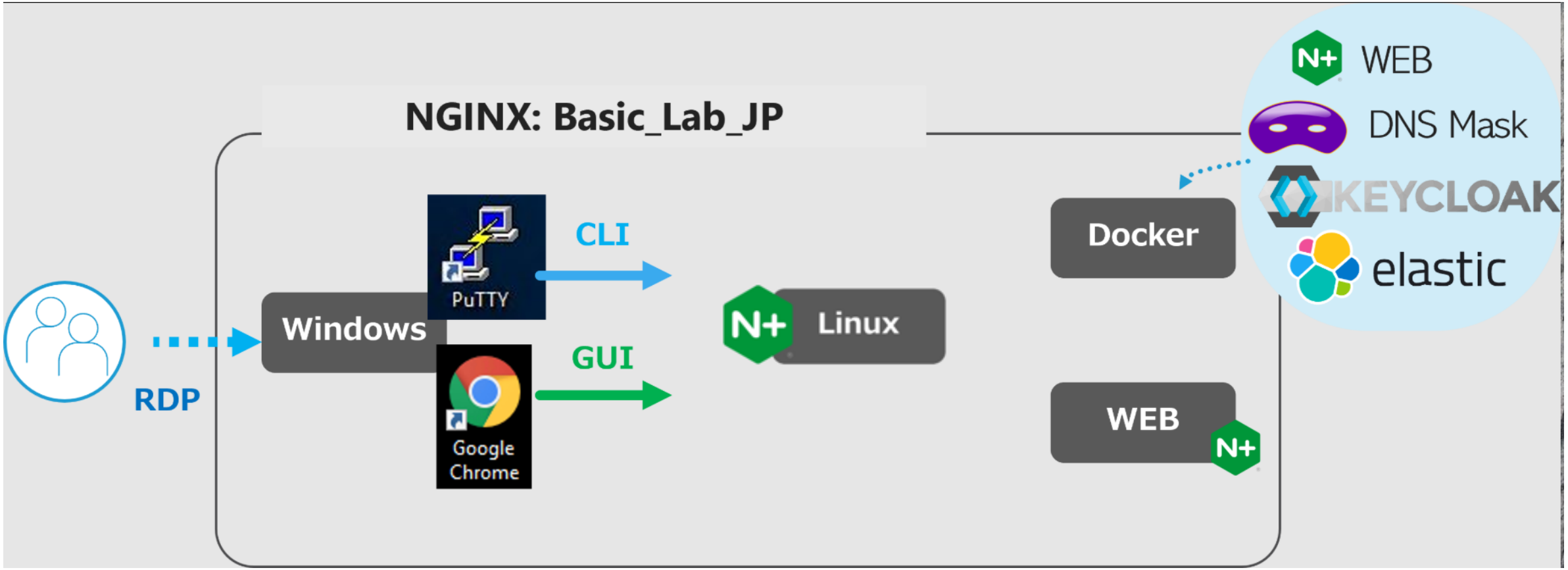
# 設定集のダウンロード

ubuntu04 のターミナルにて以下のコマンドを実行してください

```
cd ~/
git clone https://github.com/BeF5/f5j-nginx-plus-lab2-conf
```

本ハンズオンで使用する設定集がダウンロードされます。

# ハンズオン環境の構成



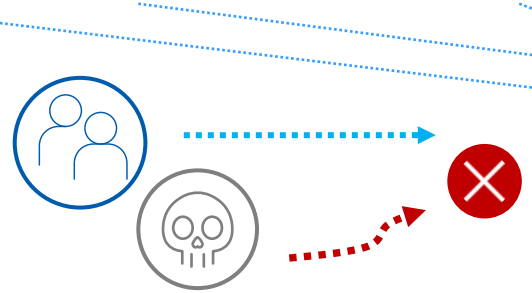
# NGINX Plusについて

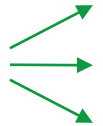
# NGINX Plusについて

## 拡張機能とサポートが追加された商用版のNGINXがNGINX Plus！

リバースプロキシ | ロードバランサー | APIGW 認証GW | WEBサーバー | キャッシュサーバー | WAF/L7DoS

Add On






Linux/BSD

CPU


オンプレミス

プライベートクラウド



パブリッククラウド


コンテナ


https://www.f5.com/pdf/product-overview/NGINX-NGINXPlus-Update-092022.pdf

# 認証・認可とは

# 認証・認可とは

## 認証（Authentication）

システムやアプリケーションなどがユーザーの身元を確認するプロセスを指す
具体的には、ID/PWや証明書で身元確認を実施、など

## 認可（Authorization）

認証されたユーザーが特定のリソースや機能にアクセスする権限を持っているかを判断するプロセスを指す
具体的には、認証されたユーザーがどのコンテンツを閲覧や編集してよいか、など

# 認証・認可とは

**本日のテーマはこちら！**

## 認証（Authentication）

システムやアプリケーションなどがユーザーの身元を確認するプロセスを指す
具体的には、ID/PWや証明書で身元確認を実施、など

## 認可（Authorization）

認証されたユーザーが特定のリソースや機能にアクセスする権限を持っているかを判断するプロセスを指す
具体的には、認証されたユーザーがどのコンテンツを閲覧や編集してよいか、など。

**この後、各認証方式をNGINX Plusで設定・実現していく設定や動作確認をハンズオントレーニングしていきます！**

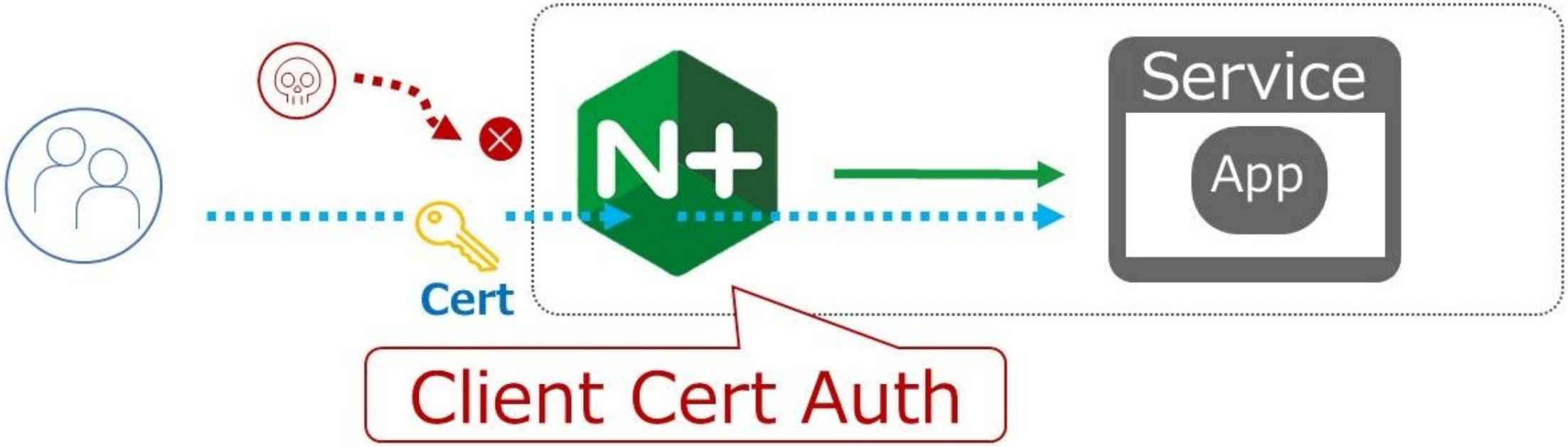
**本日のメニューはBasic認証やmTLS、JWT、OIDCを使った認証です**

# mTLS証明書認証

# mTLS証明書認証

## mTLS(mutual TLS)

クライアントとサーバーの双方が互いに証明書を提示し、相互に認証を行うことで通信の安全性を高めるプロトコル
本ハンズオンではTLS証明書を利用してmTLS認証の設定や動作確認を行う




クライアントが正しい証明書を提示していることを確認し、**問題なければ接続を許可**

# mTLS証明書認証　動作確認の流れ

- CAで利用するルート証明書を作成
- TLSを終端する際に利用するサーバー証明書を作成
- クライアント証明書認証で利用する証明書で必要となるCSRを作成
- クライアント証明書を２つ作成（CLIENT1.pemとCLIENT2.pem）
- NGINX Plusの設定確認
- クライアント証明書を提示せずに通信確認
- クライアント証明書を提示しての通信確認
- 2つ目のクライアント証明書をRevokeした際の通信確認

# TLSを終端する設定とmTLS証明書認証を行う設定

```
upstream server_group {
    zone backend 64k;

    server backend1:81;
}

server {
    listen 443 ssl;
    ssl_certificate_key conf.d/ssl/SERVER.key;
    ssl_certificate conf.d/ssl/SERVER.pem;

    ssl_client_certificate conf.d/ssl/CA.pem;
    ssl_verify_client on;

    location / {
        proxy_pass http://server_group;
    }
}
```

・待ち受けポートを443に指定しSSL/TLSを有効化
・TLSで使用する秘密鍵を指定
・TLSで使用するサーバー証明書を指定

・クライアント証明書を検証するためのCA証明書を指定
・クライアント証明書の検証を必須に指定

# クライアント証明書なしでの通信確認

curl -v --cacert ./CA.pem https://webapp.example.com --resolve webapp.example.com:443:127.0.0.1

```
< HTTP/1.1 400 Bad Request
< Server: nginx/1.21.6
< Date: Mon, 26 Sep 2022 11:01:38 GMT
< Content-Type: text/html
< Content-Length: 237
< Connection: close
<
<html>
<head><title>400 No required SSL certificate was sent</title></head>
<body>
<center><h1>400 Bad Request</h1></center>
<center>No required SSL certificate was sent</center>
<hr><center>nginx/1.21.6</center>
```

証明書を提示していないクライアントからの通信のため**アクセス拒否**

# クライアント証明書ありでの通信確認

curl -v --cacert ./CA.pem --key ./**CLIENT1.key** --cert ./**CLIENT1.pem** https://webapp.example.com --resolve webapp.example.com:443:127.0.0.1

```
> GET / HTTP/1.1
> Host: webapp.example.com
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.21.6
< Date: Mon, 26 Sep 2022 11:35:15 GMT
< Content-Type: application/octet-stream
< Content-Length: 65
< Connection: keep-alive
<
```

**証明書を提示しているクライアントからの通信のため アクセス許可**

# 証明書失効リストを確認する設定

```
upstream server_group {
    zone backend 64k;

    server backend1:81;
}

server {
   listen 443 ssl;
   ssl_certificate_key conf.d/ssl/SERVER.key;
   ssl_certificate conf.d/ssl/SERVER.pem;
   ssl_crl conf.d/ssl/CRL.pem;
   ssl_client_certificate conf.d/ssl/CA.pem;
   ssl_verify_client on;

   location / {
       proxy_pass http://server_group;
   }
}
```

・クライアント証明書の失効状態を証明書失効リスト（CRL）を参照し失効していた場合はアクセスを拒否する設定

# Revokeしたクライアント証明書での通信確認

curl -v --cacert ./CA.pem --key ./**CLIENT2.key** --cert ./**CLIENT2.pem** https://webapp.example.com --resolve webapp.example.com:443:127.0.0.1

```
* Mark bundle as not supporting multiuse
< HTTP/1.1 400 Bad Request
< Server: nginx/1.21.6
< Date: Mon, 26 Sep 2022 11:07:13 GMT
< Content-Type: text/html
< Content-Length: 215
< Connection: close
<
<html>
<head><title>400 The SSL certificate error</title></head>
<body>
<center><h1>400 Bad Request</h1></center>
<center>The SSL certificate error</center>
<hr><center>nginx/1.21.6</center>
```

有効期限が切れている証明書を提示している
クライアントからの通信のため**アクセス拒否**

# Revokeしていないクライアント証明書での通信確認

curl -v --cacert ./CA.pem --key ./**CLIENT1.key** --cert ./**CLIENT1.pem** https://webapp.example.com --resolve webapp.example.com:443:127.0.0.1

```
> GET / HTTP/1.1
> Host: webapp.example.com
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.21.6
< Date: Mon, 26 Sep 2022 11:35:15 GMT
< Content-Type: application/octet-stream
< Content-Length: 65
< Connection: keep-alive
<
```

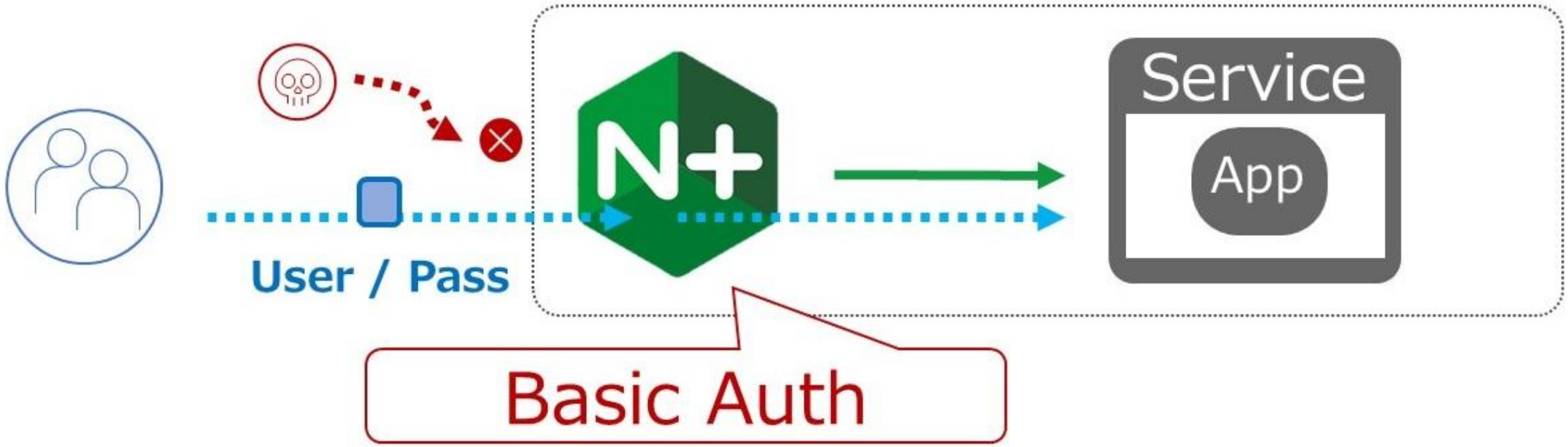
**証明書を提示しているクライアントからの通信のため アクセス許可**

# Basic認証

# Basic認証

HTTPプロトコルに組み込まれたシンプルな認証方式

クライアントが特定のコンテンツやパスにアクセスしようとする際にUserとパスワードの入力を求め、正しい認証情報であればアクセスが許可され正しくない場合はアクセスが拒否される




UserとPasswordの組み合わせを確認し、**問題なければ接続を許可**

# Basic認証　動作確認の流れ

- ユーザー、パスワードを指定せずに/authにアクセス
- user1/user1を指定して/authにアクセス

# Basic認証の設定

```
upstream server_group {
    zone backend 64k;

    server backend1:81;
}

server {
   listen 80;
   location / {
       proxy_pass http://server_group;
   }
   location /auth {
       auth_basic           "Administrator's Area";
       auth_basic_user_file conf.d/password/htpasswd;
       proxy_pass http://server_group;
   }
}
```

・/auth パスにアクセスする際に以下の設定を適用
・Basic認証を有効にし認証領域名を指定（自由に設定可）
・Basic認証時に使用するhtpasswdのファイルを指定

# ユーザー、パスワードなしでの通信確認

curl -v -s localhost/auth

```
* Mark bundle as not supporting multiuse
< HTTP/1.1 401 Unauthorized
< Server: nginx/1.21.6
< Date: Mon, 26 Sep 2022 13:27:31 GMT
< Content-Type: text/html
< Content-Length: 179
< Connection: keep-alive
< WWW-Authenticate: Basic realm="Administrator's Area"
<
<html>
<head><title>401 Authorization Required</title></head>
<body>
<center><h1>401 Authorization Required</h1></center>
<hr><center>nginx/1.21.6</center>
```

ユーザー、パスワードなしでの通信のため
**アクセス拒否**

# 正しいユーザー、パスワードでの通信確認

curl -v -s -u **user1:user1** localhost/auth

```
> GET / HTTP/1.1
> Host: webapp.example.com
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.21.6
< Date: Mon, 26 Sep 2022 11:35:15 GMT
< Content-Type: application/octet-stream
< Content-Length: 65
< Connection: keep-alive
<
```

正しいユーザー、パスワードでの通信のため
**アクセス許可**

# JWTによる通信制御

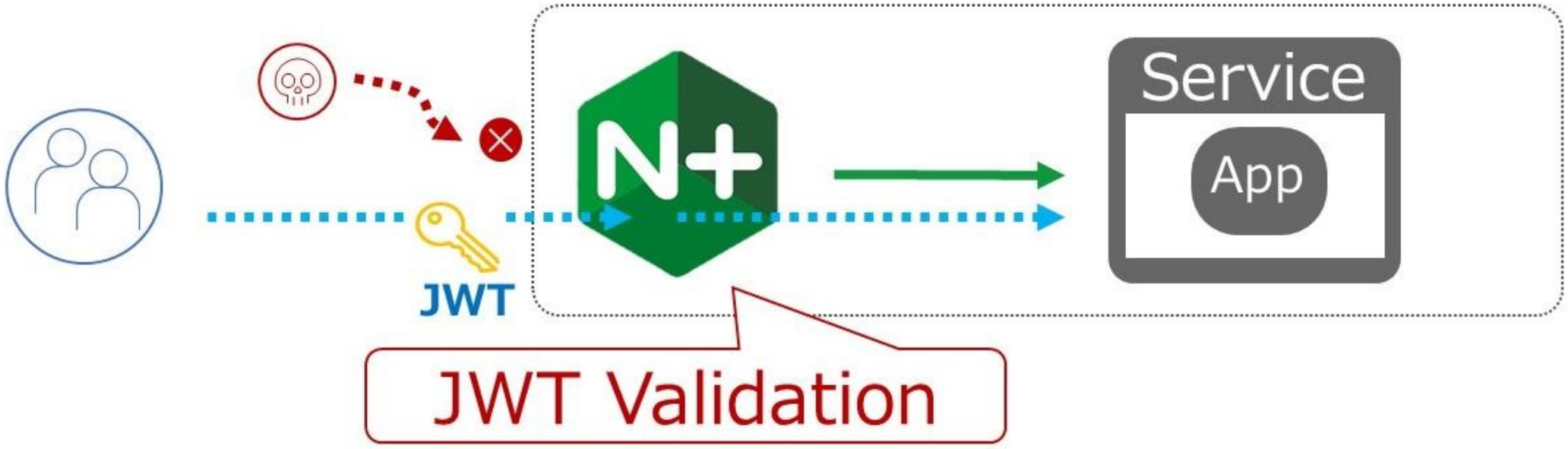
# JWTによる通信制御

**NGINX Plusで利用可能な拡張機能！**

## JWT(JSON Web Token)認証

トークンベースの認証方式

JWTはヘッダー、ペイロード、署名の3つの部分で構成され、これらを組み合わせて1つのトークンとして扱う
トークンはユーザーの認証情報や権限などの情報を含み、サーバー側でセッション情報を保持せずにユーザーの認証を行うことができる




JWK(Json Web Key)を確認し、**問題なければ接続を許可**

# JWTによる通信制御　動作確認の流れ

- JWTの確認
- JWTを含まないリクエストを送信
- JWTを含むリクエストを送信
- JWTの詳細ログを確認
- 複雑な制御

# 利用するJWTの確認

## JWK(Json Web Key)をbase64デコードした結果


```
{"keys":
   [{
      "k":"ZmFudGFzdGljand0",
      "kty":"oct",
      "kid":"0001"
   }]
}
```

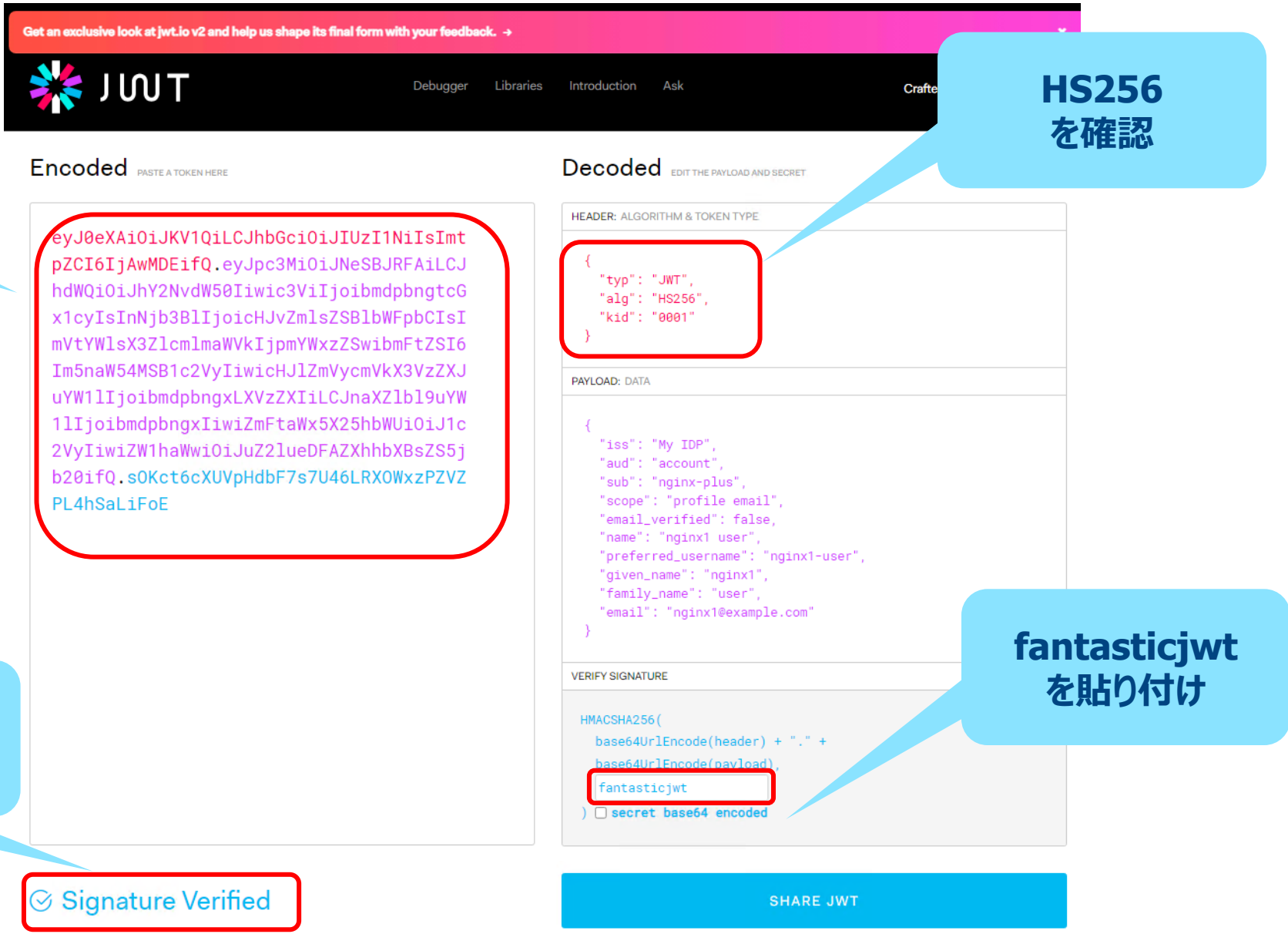

## kty “oct” で利用する Keyの内容をBase64デコードした結果

**fantasticjwt**

| Parameter | 意味 |
|---|---|
| k | k (key value) パラメータは, kty octで利用する base64url encodeされたKey文字列をもつ |
| kty | kty (key type) パラメータは, RSA や EC といった暗号アルゴリズムファミリーを示す |
| kid | kid (key ID) パラメータは特定の鍵を識別するために用いられる |

# 利用するJWTの確認

nginx1.jwtの値
を貼り付け




HS256
を確認

fantasticjwt
を貼り付け

Signature Verified
であることを確認

# JWT認証の設定

```
upstream server_group {
    zone backend 64k;

    server app-backend1:8080;
}

server {
   listen 80;
   location / {
       proxy_pass http://server_group;
   }
   location /auth {
       auth_jwt "Products API";
       auth_jwt_key_file conf.d/jwt/api_secret.jwk;
       proxy_pass http://server_group;
   }
}
```

・/auth パスにアクセスする際に以下の設定を適用
・JWT認証を有効にし認証領域名を指定（自由に設定可）
・JWTの署名を検証するための鍵が格納されたファイルを指定
**※NGINX Plusで標準搭載されているモジュール**

# JWTを含まないリクエストでの通信確認

curl -v localhost/auth

```
* Mark bundle as not supporting multiuse
< HTTP/1.1 401 Unauthorized
< Server: nginx/1.21.6
< Date: Mon, 26 Sep 2022 14:41:47 GMT
< Content-Type: text/html
< Content-Length: 179
< Connection: keep-alive
< WWW-Authenticate: Bearer realm="Products API"
<
<html>
<head><title>401 Authorization Required</title></head>
<body>
<center><h1>401 Authorization Required</h1></center>
<hr><center>nginx/1.21.6</center>
</body>
```

JWTなしでのリクエストのため
**アクセス拒否**

# JWTを含むリクエストでの通信確認

curl -v localhost/auth -H "Authorization: Bearer `cat ~/f5j-nginx-plus-lab2-conf/jwt/**nginx1.jwt**`"

```
> GET /auth HTTP/1.1
> Host: localhost
> User-Agent: curl/7.68.0
> Accept: */*
> Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6IjAwMDEifQ.eyJpc3MiOiJNeSBJRFAiLCJhdWQiOiJhY2NvdW50Iiwic3ViIjoibmdpbngtcGx1cyIsInNjb3BlIjoicHJvZmlsZSBlbWFpbCIsImVtYWlsX3ZlcmlmaWVkIjpmYWxzZSwibmFtZSI6Im5naW54MSB1c2VyIiwicHJlZmVycmVkX3VzZXJuYW1lIjoibmdpbngxLXVzZXIiLCJnaXZlbl9uYW1lIjoibmdpbngxIiwiZmFtaWx5X25hbWUiOiJ1c2VyIiwiZW1haWwiOiJuZ2lueDFAZXhhbXBsZS5jb20ifQ.sOKct6cXUVpHdbF7s7U46LRXOWxzPZVZPL4hSaLiFoE
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.21.6
< Date: Mon, 26 Sep 2022 14:47:08 GMT
< Content-Type: application/octet-stream
< Content-Length: 940
< Connection: keep-alive
```

有効なJWTを含むリクエストのため
**アクセス許可**

# JWTの詳細ログ確認用の設定確認

```
log_format jwt '$remote_addr - $remote_user [$time_local] "$request" '
               '$status $body_bytes_sent "$http_referer" "$http_user_agent" '
               '$jwt_header_alg $jwt_claim_sub $jwt_claim_scope $jwt_claim_name $jwt_claim_email';
～略～
    proxy_set_header API-Client $jwt_claim_sub;
    proxy_set_header JWT-alg $jwt_header_alg;
    proxy_set_header JWT-sub $jwt_claim_sub;
    proxy_set_header JWT-scope $jwt_claim_scope;
    proxy_set_header JWT-email $jwt_claim_email;

    access_log /var/log/nginx/access_jwt.log jwt;
    proxy_pass http://server_group;
  }
}
```

ログフォーマットを指定

・proxy_set_headerを用いてバックエンドサーバーへ転送するリクエストにJWTの情報をHTTPヘッダーとして付与
・アクセスログをjwtフォーマットに指定

# JWTを含む通信のリクエストヘッダーの確認 1

curl -s localhost/auth -H "Authorization: Bearer `cat ~/f5j-nginx-plus-lab2-conf/jwt/**nginx1.jwt**`" | jq .request.headers


```
[
  [
    "API-Client",
    "nginx-plus"
  ],
  [
    "JWT-alg",
    "HS256"
  ],
  [
    "JWT-sub",
    "nginx-plus"
  ],
  [
    "JWT-scope",
    "profile email"
  ],
  [
    "JWT-email",
    "nginx1@example.com"
  ],
  [
    "Host",
    "server_group"
  ],
  [
    "Connection",
    "close"
  ],
  [
    "User-Agent",
    "curl/7.68.0"
  ],
  [
    "Accept",
    "*/*"
  ],
  [
    "Authorization",
    "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6IjAwMDEifQ.eyJpc3MiOiJNeSBJRFAiLCJhdWQiOiJhY2NvdW50Iiwic3ViIjoibmdpbngtcGx1cyIsInNjb3BlIjoicHJvZmlsZSBlbWFpbCIsImVtYWlsX3ZlcmlmaWVkIjpmYWxzZSwibmFtZSI6Im5naW54MSB1c2VyIiwicHJlZmVycmVkX3VzZXJuYW1lIjoibmdpbngxLXVzZXIiLCJnaXZlbl9uYW1lIjoibmdpbngxIiwiZmFtaWx5X25hbWUiOiJ1c2VyIiwiZW1haWwiOiJuZ2lueDFAZXhhbXBsZS5jb20ifQ.sOKct6cXUVpHdbF7s7U46LRXOWxzPZVZPL4hSaLiFoE"
  ]
]
```

# JWTを含む通信のリクエストヘッダーの確認２

curl -s localhost/auth -H "Authorization: Bearer `cat ~/f5j-nginx-plus-lab2-conf/jwt/**nginx2.jwt**`" | jq .request.headers


```
[
  [
    "API-Client",
    "nginx-plus"
  ],
  [
    "JWT-alg",
    "HS256"
  ],
  [
    "JWT-sub",
    "nginx-plus"
  ],
  [
    "JWT-scope",
    "profile email"
  ],
  [
    "JWT-email",
    "nginx2@example.com"
  ],
  [
    "Host",
    "server_group"
  ],
  [
    "Connection",
    "close"
  ],
  [
    "User-Agent",
    "curl/7.68.0"
  ],
  [
    "Accept",
    "*/*"
  ],
  [
    "Authorization",
    "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6IjAwMDEifQ.eyJpc3MiOiJNeSBJRFAiLCJhdWQiOiJhY2NvdW50Iiwic3ViIjoibmdpbngtcGx1cyIsInNjb3BlIjoicHJvZmlsZSBlbWFpbCIsImVtYWlsX3ZlcmlmaWVkIjpmYWxzZSwibmFtZSI6Im5naW54MiB1c2VyIiwicHJlZmVycmVkX3VzZXJuYW1lIjoibmdpbngyLXVzZXIiLCJnaXZlbl9uYW1lIjoibmdpbngyIiwiZmFtaWx5X25hbWUiOiJ1c2VyIiwiZW1haWwiOiJuZ2lueDJAZXhhbXBsZS5jb20ifQ.CVUH7upnT5n2yzLnlGhNTMKL_Ev6yJfj8FHSFC5v5ME"
  ]
]
```

# ログの確認

127.0.0.1 - - [27/Sep/2022:00:04:41 +0900] "GET /auth HTTP/1.1" 200 1079 "-" "curl/7.68.0" **HS256 nginx-plus profile email nginx1 user nginx1@example.com**

127.0.0.1 - - [27/Sep/2022:00:04:48 +0900] "GET /auth HTTP/1.1" 200 1079 "-" "curl/7.68.0" **HS256 nginx-plus profile email nginx2 user nginx2@example.com**

👆太字の箇所がログフォーマットで指定したJWT関連の変数の値

**'$jwt_header_alg $jwt_claim_sub $jwt_claim_scope $jwt_claim_name $jwt_claim_email';**

# 複雑な制御の設定確認

```
limit_req_zone $jwt_claim_sub zone=1rpm_per_client:1m rate=1r/m;
～略～
map $jwt_claim_scope $jwt_upstream {
    ~group1 "slow_group";
    ~group2 "slow_group";
    default default_group;
}
～略～
server {
    listen 80;
    location / {
        proxy_pass http://$jwt_upstream;
    }
    location /auth {
        auth_jwt "Products API";
        auth_jwt_key_file conf.d/jwt/api_secret.jwk;
        access_log /var/log/nginx/access_jwt.log jwt;

        limit_req zone=1rpm_per_client;
        proxy_pass http://$jwt_upstream;
    }
}
```

・$jwt_claim_subをKeyとしたリクエストレート制限
各クライアントごとに1分間に1リクエストの制限
今回は$jwt_claim_subの値は**nginx-plus**

・$jwt_claim_scopeの値がgroup1、group2、それ以外の場合
それぞれで$jwt_upstreamの変数を決定する
group1、group2は"slow_group"　app-backend1:8080
それ以外はdefault_group　app-backend2:8080

・/へのアクセス時に$jwt_upstreamにプロキシ

・/authへのアクセス時に$jwt_upstreamにプロキシ

# nginx3.jwtで1回目/分のアクセスを実施

curl -s localhost/auth -H "Authorization: Bearer `cat ~/f5j-nginx-plus-lab2-conf/jwt/**nginx3.jwt**`" | jq .request.headers


```
[
  [
    "Host",
    "slow_group"
  ],
  [
    "Connection",
    "close"
  ],
  [
    "User-Agent",
    "curl/7.68.0"
  ],
```

```
  [
    "Accept",
    "*/*"
  ],
  [
    "Authorization",
    "Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6IjAwMDEifQ.eyJpc3MiOiJNeSBJRFAiLCJhdWQiOiJ
hY2NvdW50Iiwic3ViIjoibmdpbngtcGx1cyIsInNjb3BlIjoicHJvZmlsZSBlbWFpbCBncm91cDIiLCJlbWF
pbF92ZXJpZmllZCI6ZmFsc2UsIm5hbWUiOiJuZ2lueDMgdXNlciIsInByZWZlcnJlZF91c2VybmFtZSI6I
m5naW54My11c2VyIiwiZ2l2ZW5fbmFtZSI6Im5naW54MyIsImZhbWlseV9uYW1lIjoidXNlciIsImVtY
WlsIjoibmdpbngzQGV4YW1wbGUuY29tIn0.CGa2fDJFiTJwlNgqW6IdCENu_Re0gkPTaww-glCHckM"
  ]
]
```


1回目/分のリクエストのため
**アクセス許可**

# nginx1.jwtで2回目/分のアクセスを実施

curl -s localhost/auth -H "Authorization: Bearer `cat ~/f5j-nginx-plus-lab2-conf/jwt/**nginx1.jwt**`"

```
<html>
<head><title>503 Service Temporarily Unavailable</title></head>
<body>
<center><h1>503 Service Temporarily Unavailable</h1></center>
<hr><center>nginx/1.21.6</center>
</body>
</html>
```

2回目/分のリクエストのため
**アクセス拒否**

# ログの確認

**1回目のアクセス**
**$jwt_claim_scope ⇒profile email group2**
**$jwt_claim_sub ⇒nginx-plus**
**$jwt_upstream ⇒slow_group**
**レスポンスコード200、$jwt_claim_scope に group2 が含まれているため、$jwt_upstreamがslow_groupとなる**

127.0.0.1 - - [27/Sep/2022:09:38:00 +0900] "GET /auth HTTP/1.1" **200** 948 "-" "curl/7.68.0" **profile email group2 nginx3** user nginx3@example.com **nginx-plus slow_group**

**2回目のアクセス**
**$jwt_claim_scope ⇒profile email**
**$jwt_claim_sub ⇒nginx-plus**
**$jwt_upstream ⇒default_group**
**レスポンスコード503、$jwt_claim_scope に group2 が含まれていないため、$jwt_upstreamがdefault_groupとなる**

127.0.0.1 - - [27/Sep/2022:09:38:07 +0900] "GET /auth HTTP/1.1" **503** 197 "-" "curl/7.68.0" **profile email nginx1** user nginx1@example.com **nginx-plus default_group**

1回目と2回目のアクセスともに、$jwt_claim_subが**nginx-plus**となっている
この値がRate LimitのKeyとなっているため、分間2回の通信で2回目アクセスが**Rate Limit**で**拒否**されています

# エラーログの確認

2022/09/27 09:38:07 [**error**] 1845#1845: *17 limiting requests, excess: 0.886 by zone "**1rpm_per_client**", client: 127.0.0.1, server: , request: "GET /auth HTTP/1.1", host: "localhost"

2回目のアクセスが1rpm_per_clientでRequest Limitの設定に該当した結果エラーとなったことがわかる

```
limit_req_zone $jwt_claim_sub zone=1rpm_per_client:1m rate=1r/m;
```

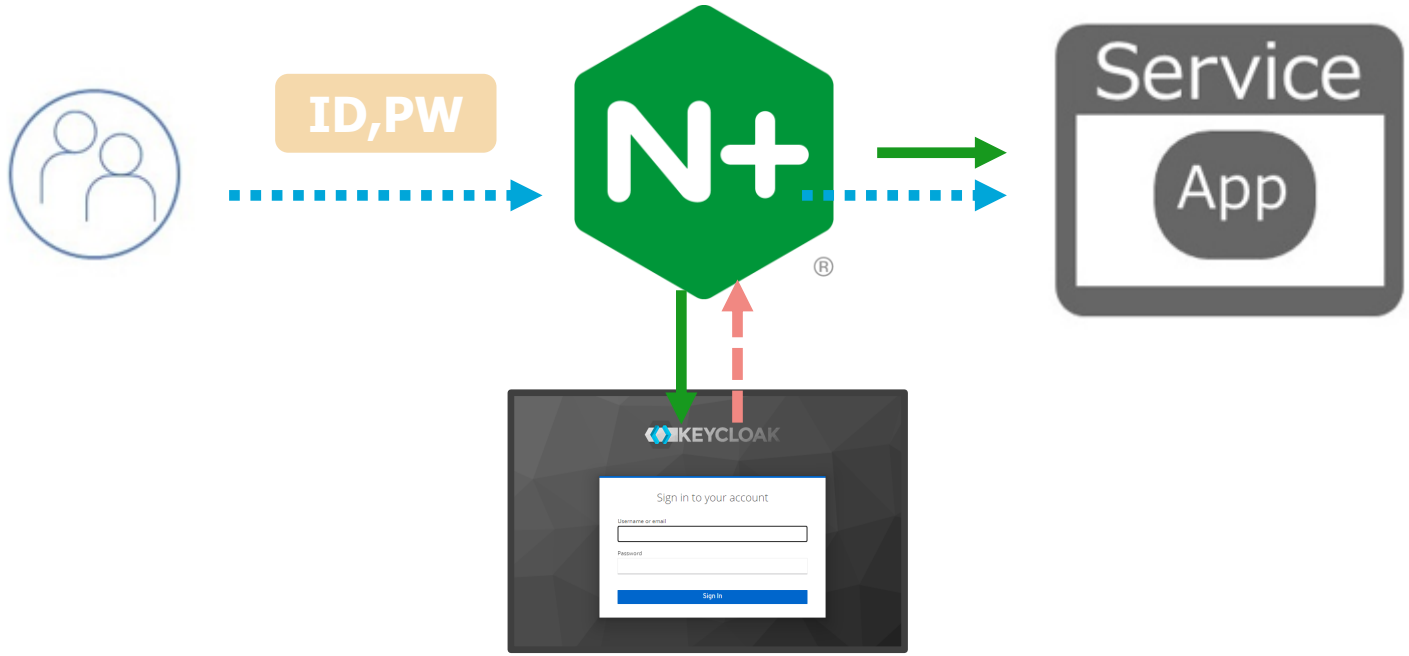
# OIDCによる通信制御

# OIDCによる通信制御

**NGINX Plusで利用可能な拡張機能！**
※NGINX OSSでも一部利用可能だが実装難易度高

## OIDC

OIDC（OpenID Connect）は、OAuth 2.0プロトコルを基盤とした認証プロトコルであり、ユーザーのアイデンティティを安全かつシンプルに確認するための標準仕様
OIDCを利用することで、異なるサービス間でユーザーが同一のログイン情報を使用できるSSO（シングルサインオン）を実現することができる

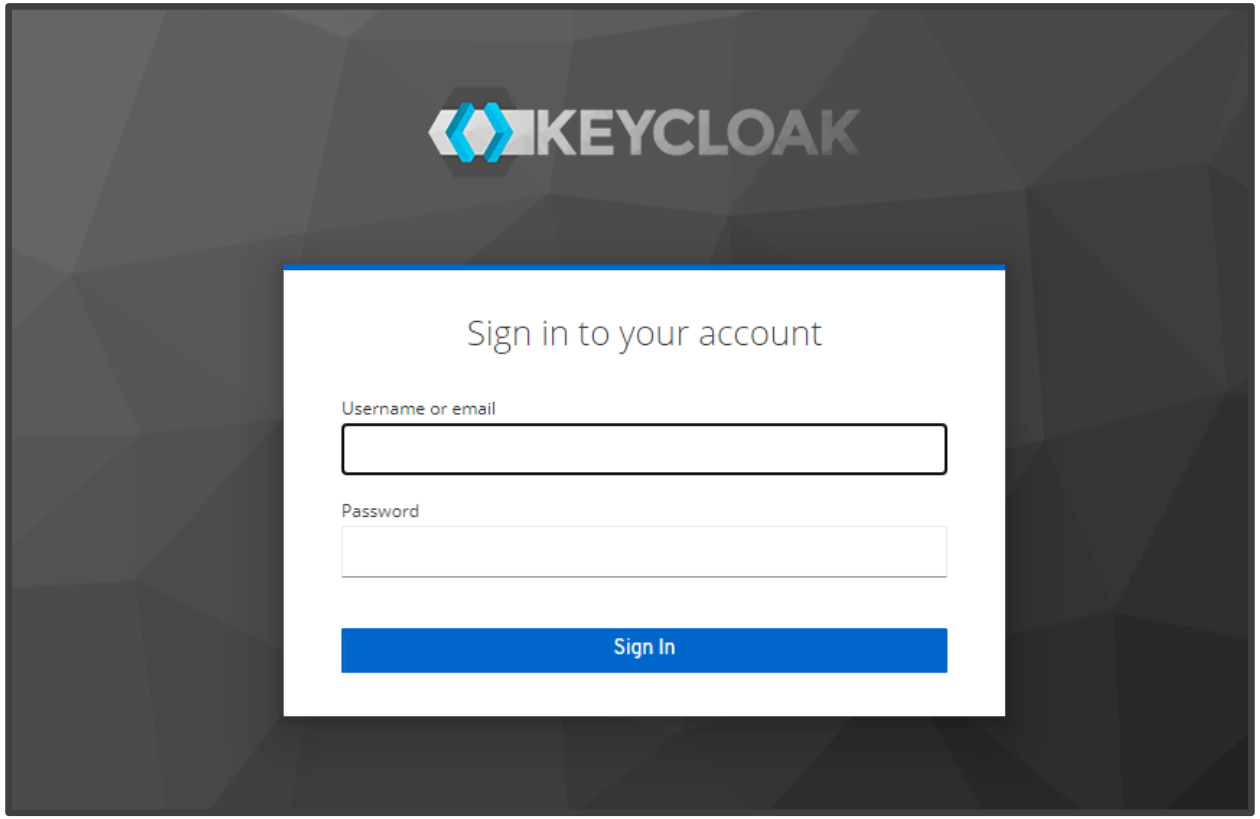



ID,PWを確認し、**問題なければ接続を許可**

# Keycloakとは

## Keycloak

オープンソースのアイデンティティやアクセス管理（IAM）ソフトウェアであり、SSO（シングルサインオン）やAPIアクセスの認証・認可制御を提供している
Open ID ConnectやOAuth2.0、SAMLなどのプロトコルに対応している



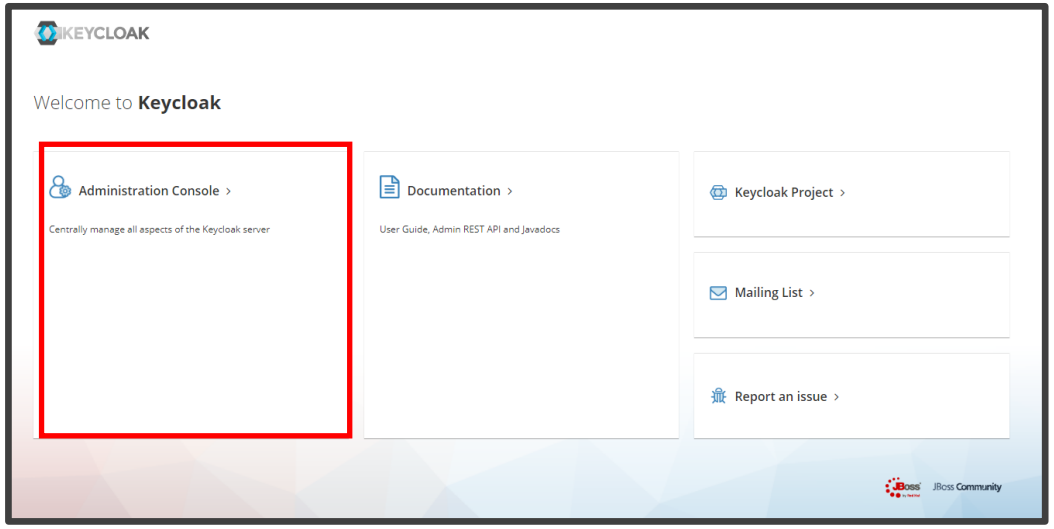
# OIDCよる通信制御　動作確認の流れ

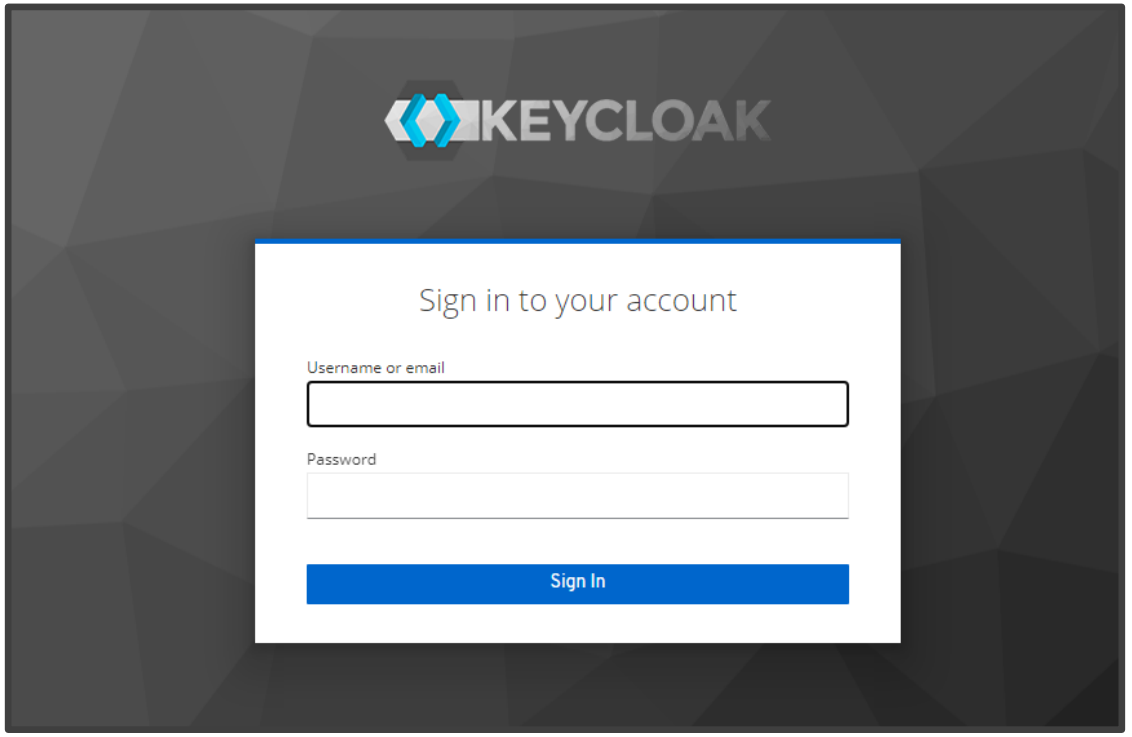
- Keycloakでユーザーを作成
- NGINX PlusでNJSモジュールのインストール
- OIDCに必要となる情報をKeycloakから取得
- 設定確認
- SSOの動作確認

# Keycloakでユーザー作成

ログイン画面にて以下の情報でログインを実施
Username：admin
Password：admin

http://10.1.1.5:8081/にアクセスを実施




ユーザー作成はテキストに従って実施します
Valid Redirect URIsは**http://10.1.1.11:80/_codexch**

# OIDCの設定確認

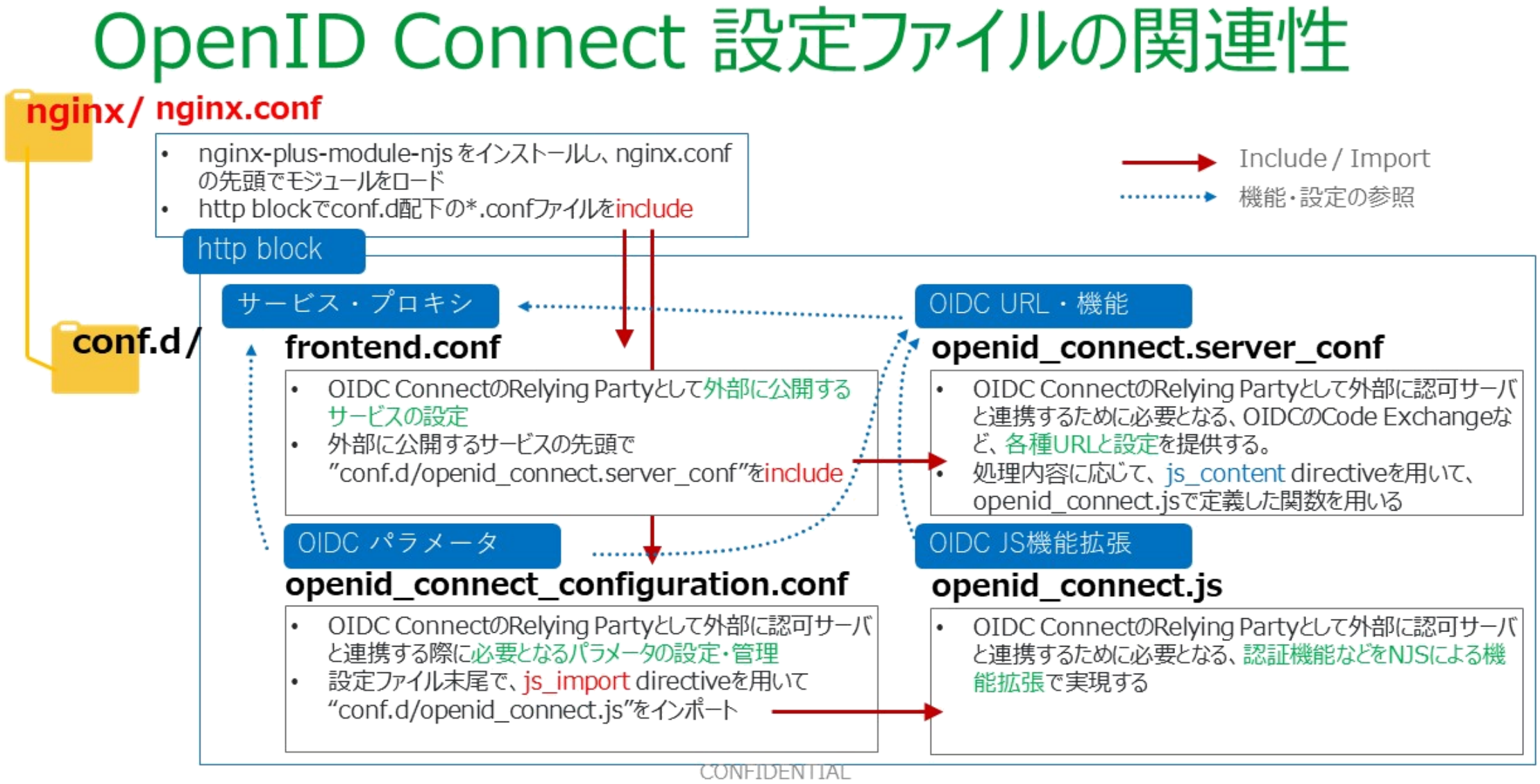
```
upstream my_backend {
    zone my_backend 64k;
    server backend1:81;
～略～
server {
    include conf.d/openid_connect.server_conf;
～略～
 location / {
        proxy_hide_header "Content-Type";
        add_header 'Content-Type' 'text/html';

        # This site is protected with OpenID Connect
        auth_jwt "" token=$session_jwt;
        error_page 401 = @do_oidc_flow;

        auth_jwt_key_file $oidc_jwt_keyfile;
～略～
```

・転送先を指定

・OIDCに必要となる各種Pathを指定した別の設定ファイルを読み込み

・OIDCのフローに従って正しくJWT Tokenを取得したクライアントのアクセスを評価

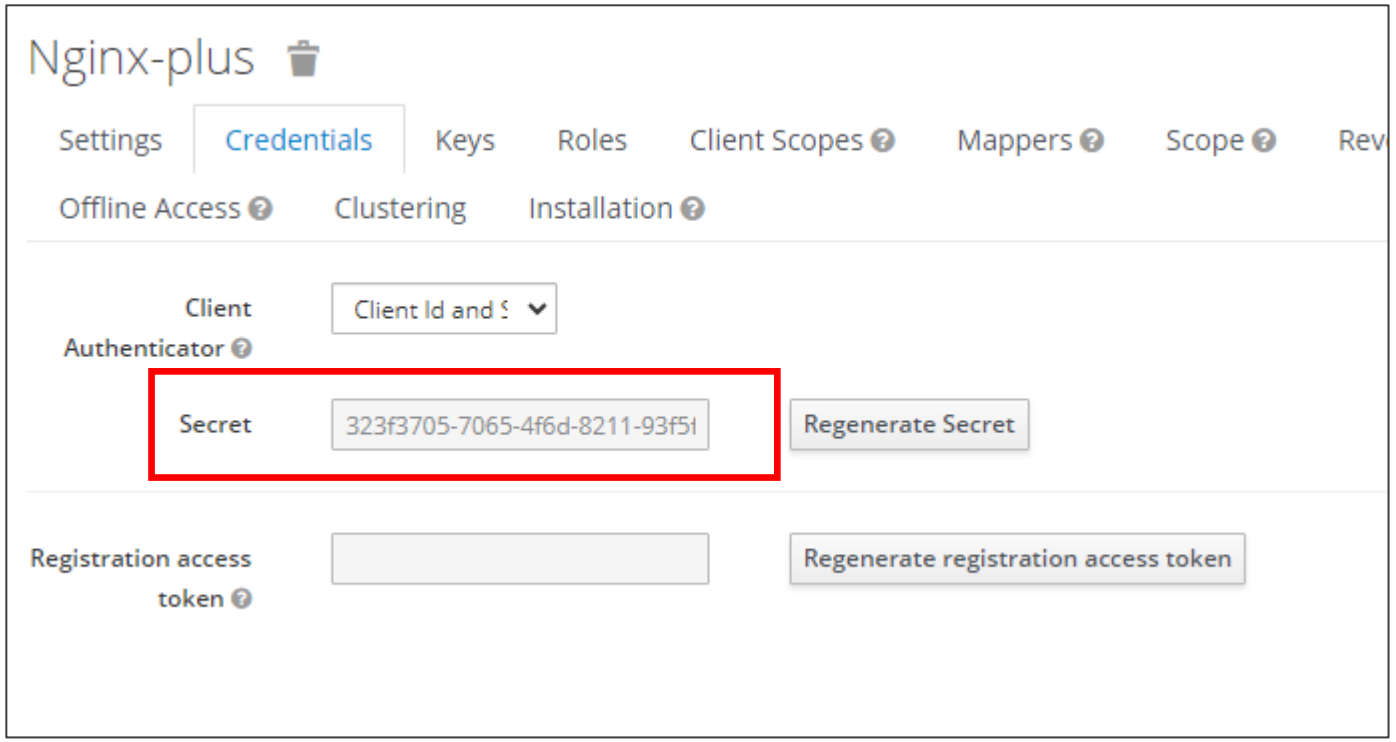
# OIDC設定ファイルについて



# Keycloakと連携するためのID/シークレットを設定

sed -i -e 's/my-client-id/nginx-plus/g' ~/nginx-openid-connect/openid_connect_configuration.conf
sed -i -e 's/my-client-secret/<**Client Secret**>/g' ~/nginx-openid-connect/openid_connect_configuration.conf

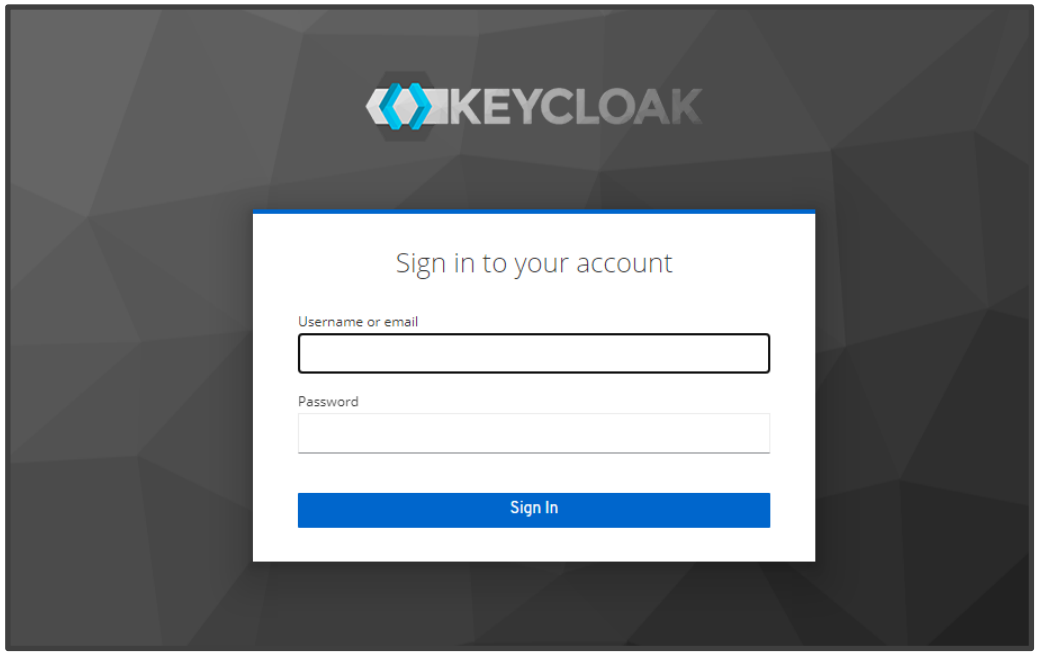
**<Client Secret>部分を削除して、Keycloakからコピーした値を貼り付ける**
**例）323f3705-7065-4f6d-8211-93f5feec5ccc**



# 動作確認

## Keycloakで作成したUsername,PWでログイン

Username: nginx-user
Password:test




入力されたUsername,PasswordをKeycloakに情報提供し認証されたためbackend1のコンテンツが問題なく表示される！

# まとめ

# まとめ

以下についてご紹介しました。

- NGINX Plus について
- mTLS証明書認証
- Basic認証
- JWTによる通信制御
- OIDCによる通信制御

# NGINXの情報発信しています！

## NGINXがまるっと分かる！ブログ更新中！

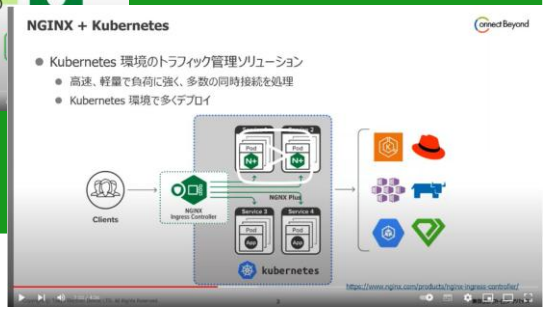
https://cn.teldevice.co.jp/blog/search/?q=NGINX

ブログ

YouTube

## NGINXを簡単解説！

F5 NGINX - YouTube






東京エレクトロン デバイス