nAxiom

Wdrożenie nAxiom z obrazów Docker

Wersja nAxiom: 1.11.3

Spis treści

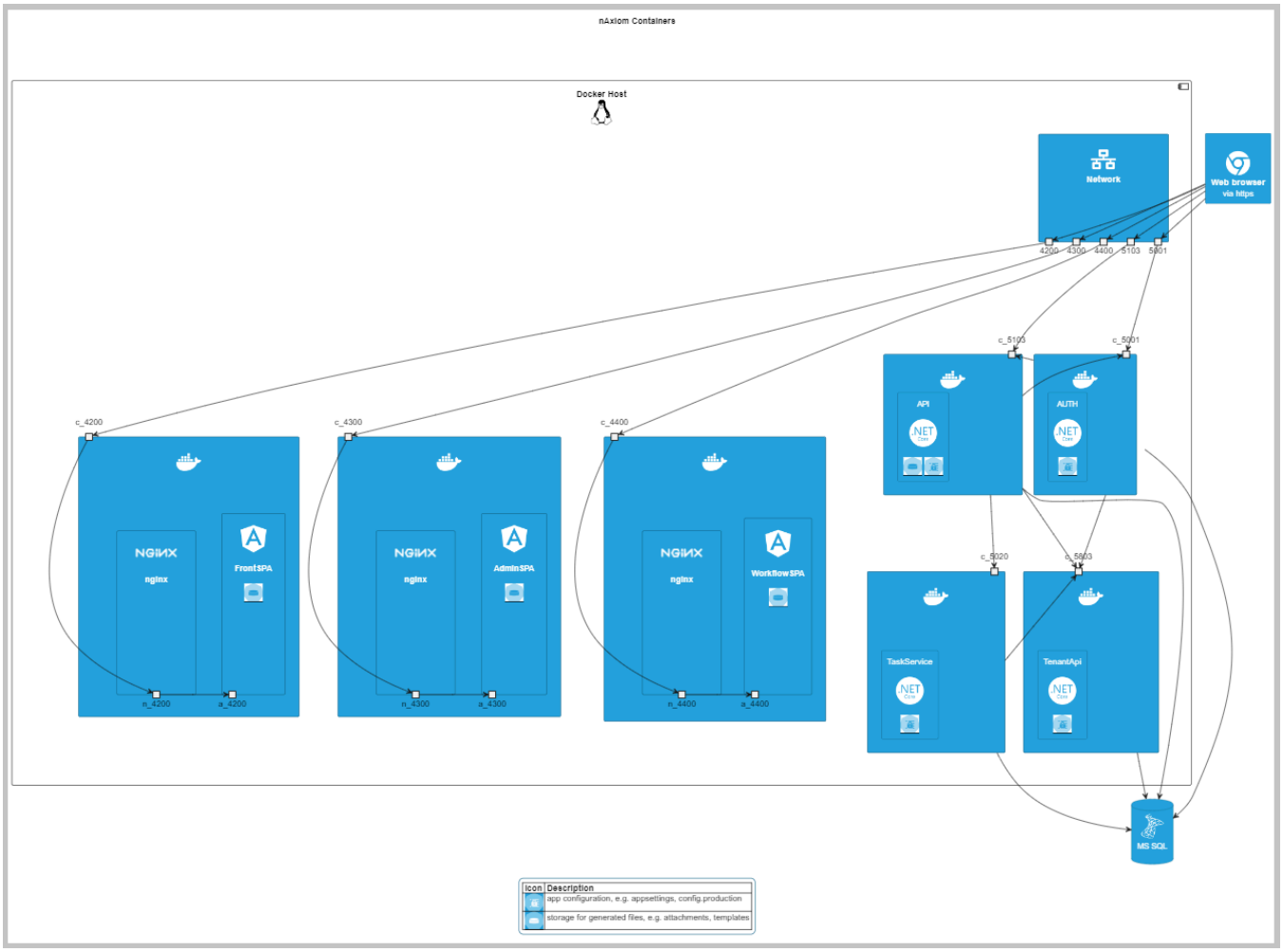
1. Wprowadzenie

Platforma nAxiom jest dostępna jako zestaw obrazów Docker dla systemu Linux (architektura x86-64). Obrazy umożliwiają wdrożenie nAxiom jako zestawu kontenerów Docker lub zestawu kontenerów konfigurowanych w klastrze Kubernetes. (Odniesienia do platformy Kubernetes należy traktować jako poglądowe; możliwe jest użycie dowolnej platformy orkiestracji kontenerów). Uruchomienie nAxiom z

obrazów w systemie Windows (w celach poglądowych/demonstracyjnych) wymaga użycia Windows Subsystem for Linux (WSL 2) — na potrzeby takiego wdrożenia opracowano specjalny skrypt, który automatyzuje wdrożenie, jego opis zawiera Dodatek: Wdrożenie w Windows z użyciem skryptu startowego.
Wdrożenie było testowane na dystrybucjach Debian i Ubuntu, ale w ogólności obsługiwane są dowolne dystrybucje, o ile zostaną uwzględnione indywidualne różnice (powłoki, instalowane domyślnie pakiety itp.).
Zestaw obrazów nAxiom jest dostarczany jako plik archiwum *tar* i obejmuje następujące serwisy (w nawiasie podano typ serwisu):

- auth (dotnet) - uwierzytelnianie użytkowników
- front (angular) - obsługa aplikacji biznesowych przez użytkowników końcowych
- admin (angular) - tworzenie aplikacji biznesowych przez konsultantów
- tenant-admin (angular) - zarządzanie tenantami w środowisku *multi tenant*
- api (dotnet) - obsługa żądań do bazy danych wysyłanych z serwisu *admin* i *front*
- tenant-api (dotnet) - obsługa żądań do bazy danych wysyłanych z serwisu *tenant-admin*
- task-service (dotnet) - obsługa zadań cyklicznych, wysyłki e-mail, śledzenia terminów
- workflow (angular) - definiowanie diagramów procesów biznesowych
- documentation (dotnet) - obsługa pomocy kontekstowej i funkcji generowania dokumentacji aplikacji biznesowych
- public-api (dotnet) - dostęp do publicznego interfejsu API platformy nAxiom
- mobile-api (dotnet) - obsługa aplikacji mobilnej nAxiom
- syncfusion-api (dotnet) - obsługa generowania, szyfrowania i podglądu plików PDF
- telerik-reports (dotnet) - obsługa definiowania, generowania i podglądu raportów Telerik

W tej wersji nAxiom nie są dostępne serwisy Crystal Reports, OCR i ePUAP, w związku z czym obsługiwane przez nie funkcjonalności również nie są dostępne ani w aplikacji *AdminSPA* ani przez żądania do *PublicAPI*.
Obrazy są oznaczone tagiem z numerem wersji nAxiom.
Poniższy diagram przedstawia połączenia wybranych kontenerów nAxiom.



Przykładowy zestaw kontenerów nAxiom

Ze względów bezpieczeństwa kontenery serwisów nAxiom są uruchamiane z poziomu użytkownika nieuprzywilejowanego (*rootless*), który ma dostęp tylko do katalogu roboczego kontenera (*/home/app* dla serwisów dotnetowych). Dodatkowym zabezpieczeniem jest mechanizm plików chronionych w Dockerze, który usuwa pliki z poufnymi danymi z kontenera po ich wczytaniu podczas uruchamiania. W Kubernetes zaleca się dystrybucję plików z danymi poufnymi, np. hasłami, jako obiektów *Secret*. Ponadto komunikacja między kontenerami jest szyfrowana.

Zestaw kontenerów nAxiom może korzystać z serwera nginx (również uruchamianego w kontenerze). Także serwer bazy danych może działać w kontenerze. Ponadto można używać dodatkowego oprogramowania do wyszukiwania pełnotekstowego (Elasticsearch), obsługi pamięci podręcznej (Redis), agregacji logów (Zabbix/EFK) i innych.

2. Procedura podstawowa

Obraz każdego serwisu zawiera plik z domyślnymi ustawieniami konfiguracyjnymi (*appsettings.json* dla serwisów typu dotnet i *config.json* dla serwisów typu angular). W razie potrzeby ustawienia te można modyfikować, używając własnych plików konfiguracyjnych, wczytywanych do kontenera w trakcie uruchamiania. Minimalne wymagane czynności konfiguracyjne do uruchomienia pełnego zestawu kontenerów nAxiom to:

- przygotowanie pliku ustawień środowiskowych
- wygenerowanie i zainstalowanie certyfikatów
- zdefiniowanie parametrów połączenia z serwerem bazy danych
- załadowanie obrazów do repozytorium (`docker load`)
- utworzenie pierwszego tenanta
- uruchomienie nAxiom jako zestawu kontenerów (`docker compose`) lub klastra.
- dodatkowo należy skonfigurować reguły dla serwera proxy (obiekty *ingress* w przypadku klastra Kubernetes lub *route* w przypadku klastra Openshift)

2.1. Katalog główny wdrożenia

W niniejszym dokumencie założono, że do wdrożenia używany jest utworzony w systemie plików hosta *katalog główny wdrożenia*, w którym znajdują się niezbędne pliki. W dalszej części używany będzie katalog */home/naxadmin/nax-deploy*.
Struktura podkatalogów w głównym katalogu wdrożenia może wyglądać następująco:

- *certs*: certyfikaty używane do komunikacji między serwisami oraz do komunikacji zewnętrznej. Muszą to być certyfikaty zaufanych głównych urzędów certyfikacji.
- *configs*: pliki konfiguracyjne poszczególnych serwisów. Dla każdego serwisu należy utworzyć podkatalog o nazwie takiej jak nazwa serwisu (np. *auth*, *admin* itd.). Pliki serwisów dotnetowych mają nazwę *appsettings-custom.json*, a dla serwisów angularowych *config-custom.json*. Dodatkowo katalog każdego serwisu może zawierać plik *nlog-custom.config* z niestandardową konfiguracją logowania danego serwisu.
- *scripts*: skrypty wykonywane w trakcie uruchamiania każdego serwisu (w podkatalogach dla każdego serwisu); obecnie plik skryptu musi mieć nazwę *custom-entrypoint.sh*
- *secrets*: chronione pliki własnych ustawień konfiguracyjnych dla poszczególnych serwisów (w osobnych podkatalogach) zawierające poufne dane, np. dane logowania do bazy danych.
- *nginx*: pliki konfiguracyjne serwisów nAxiom dla serwera proxy nginx używane w skryptach *docker-compose*.

- *mssql*: pliki konfiguracyjne kontenera SQL Server; katalog do zapisu plików bazy danych.

> Uwaga: jeśli katalog *mssql* zostanie utworzony w systemie Linux, konieczne jest ustawienie dla niego pełnych uprawnień dla wszystkich (`chmod 777 -R /path/mssql`).

Na potrzeby tego opisu założono ponadto, że w katalogu głównym wdrożenia zostanie także umieszczone archiwum *tar* z obrazami serwisów nAxiom, pliki *docker-compose* używane do uruchamiania kontenerów nAxiom oraz plik *docker.env* z ustawieniami środowiska.

2.2. Ustawienia środowiska

Plik *docker.env* zawiera zmienne „środowiskowe" używane do konfiguracji wdrożenia kontenerów nAxiom. Ustawione w tym pliku wartości są używane podczas uruchamiania kontenerów poleceniem `docker compose`. Plik zawiera następujące ustawienia:

```
# tag obrazów serwisów nAxiom, domyślnie "latest"
NAX_VERSION="latest"

# katalog główny wdrożenia, zawiera wszystkie dane konfiguracyjne
NAX_ROOT_CONF_DIR="/home/naxadmin/nax-deploy"
# nazwa pliku certyfikatu (bez rozszerzenia) używanego przez kontenery
# nAxiom do komunikacji wewnętrznej
NAX_ROOT_CERTIFICATE="d4r-internal"

# ustawienia SQL Server uruchamianego w kontenerze
# tag obrazu, dla Azure SQL Edge - "latest"
NAX_MSSQL_VERSION="2019-latest"
# katalog z danymi konfiguracyjnymi mssql
NAX_MSSQL_DATA_DIR="/home/naxadmin/nax-deploy/mssql"
# hasło do instancji bazy danych mssql
NAX_MSSQL_PASSWORD="p@sswOrd"

# ustawienia serwera proxy (nginx)
# tag obrazu nginx
NAX_NGINX_VERSION="latest"
# nazwa pliku certyfikatu (bez rozszerzenia) używanego w połączeniach z nginx
```

```
NAX_NGINX_CERTIFICATE="d4r-proxy"

# nazwa domeny
DOMAIN_NAME="nax-deploy.local"
# profil wdrożenia: DOMAIN / PROXY / K8S
DEPLOY_TYPE="PROXY"
```

W pliku można określić między innymi profil wdrożenia:

- *DOMAIN*: każdy serwis jest dostępny pod adresem URL *https://naxiom-domain:port*, gdzie *port* to unikalny numer portu serwisu.
- *PROXY*: każdy serwer jest dostępny pod adresem URL *https://naxiom-domain/service-name*, gdzie *service-name* to nazwa serwisu; jako serwer proxy jest używany serwer *nginx* (działający w kontenerze).
- *K8S*: konfiguracja serwisów jest dostosowana do uruchomienia w klastrze Kubernetes.

W zależności od wartości tej zmiennej, kontenery zostają uruchomione z wersją pliku konfiguracyjnego odpowiednią dla danego profilu wdrożenia.

2.2.1. Klaster Kubernetes

Dla profilu wdrożenia *K8S* kontenery serwisów dotnetowych są uruchamiane z użyciem szablonu pliku *appsettings.json*, który dla serwisu *api* wygląda następująco:

```
{
  "AppConfiguration": {
    "FrontAppUrl": "https://${DOMAIN_NAME}/front",
    "IdentityServerUrl": "https://auth-service.${DEPLOY_NAMESPACE}:5001",
    "TaskServiceServerUrl": "https://task-service-service.${DEPLOY_NAMESPACE}:5020",
    "TelerikReportsServerUrl": "https://telerik-reports-service.${DEPLOY_NAMESPACE}:5503",
    "TenantsApiUrl": "https://tenant-api-service.${DEPLOY_NAMESPACE}:5803",
    "PublicApiServerUrl": "https://public-api-service.${DEPLOY_NAMESPACE}:5203",
    "SyncfusionApiServerUrl": "https://syncfusion-api-service.${DEPLOY_NAMESPACE}:5001",
    "OriginConfiguration": {
      "IsHostedOnOnePort": true,
      "OriginUrl": "https://${DOMAIN_NAME}"
    },
    "MultiTenancyConfig": {
      "LoadTenantConfigurationFromService": true,
```

```
      "BaseUrl": "https://${DOMAIN_NAME}"
    }
  }
}
```

W tym typie wdrożenia kluczowe jest, aby w momencie definiowania obiektów *Deployment* dla poszczególnych serwisów nAxiom korzystać z następujących nazw obiektów *Services* dla podów:

- tenant-api-service
- auth-service
- api-service
- front-service
- admin-service
- tenant-admin-service
- workflow-service
- public-api-service
- mobile-api-service
- telerik-reports-service
- documentation-service
- task-service-service
- syncfusion-api-service

Związane jest to z zapewnieniem zgodności adresów URL wykorzystywanych do komunikacji wewnętrznej, która bazuje na nazwie obiektu *Service* dla poda, np.
https://auth-service.${DEPLOY_NAMESPACE}:5001
(*auth-service* to nazwa obiektu *Service* dla poda z kontenerem serwisu *auth*). W przypadku skorzystania z innych nazw dla obiektów *Service*, należy odpowiednio zdefiniować te adresy wykorzystując do tego pliki *appsetings-custom.json*.

2.3. Certyfikaty

Kontenery nAxiom wykorzystują do komunikacji (wewnętrznej i zewnętrznej) protokół *https* i w związku z tym wymagają odpowiednich certyfikatów. Na potrzeby wdrożenia lokalnego można użyć certyfikatów *self signed*. Aby jednak w takim wdrożeniu można było obsługiwać tryb *multi tenancy*, potrzebne są certyfikaty typu *wildcard*, których nie można utworzyć dla domeny TLD, takiej jak *localhost*. To pociąga za sobą konieczność utworzenia nazw domenowych dla wdrożenia nAxiom.
Certyfikaty SSL klienta muszą zawierać nazwy DNS odpowiadające nazwom serwisów. Ponadto certyfikaty muszą zostać wczytane do kontenerów w następujących lokalizacjach:

- obrazy dotnetowe
    - .pem/.crt: /usr/local/share/ca-certificates/certificate.crt
    - .key: /root/.aspnet/https/certificate.key (klucz jest usuwany w momencie odczytania go przez aplikację podczas uruchamiania kontenera)
- obrazy angularowe:
    - .pem/.crt: /usr/local/share/ca-certificates/certificate.crt
    - .key: /root/.node/https/certificate.key

Odpowiadają za to odpowiednie wpisy w pliku *docker-compose-volumes.yml*.

2.3.1. Modyfikacja pliku *hosts*

W dalszej części tego artykułu będzie używana nazwa domenowa *nax-deploy.local*. W związku z tym wymagane jest dodanie odpowiednich wpisów do pliku *hosts* (*/etc/hosts* lub *%windir%\System32\drivers\etc\hosts*):

```
127.0.0.1       nax-deploy.local
127.0.0.1       tenant1.nax-deploy.local
127.0.0.1       tenant2.nax-deploy.local
```

wpisy z przedrostkiem *tenant...* definiują nazwy domenowe nAxiom do wykorzystania w środowisku z dwoma tenenatami (każdy tenant wymaga osobnego wpisu).

2.3.2. Certyfikat dla komunikacji wewnętrznej

Konfiguracja tego certyfikatu musi uwzględniać nazwy DNS podane w sekcji *[alt_names]*. Przykładowy plik konfiguracyjny umożliwiający wygenerowanie certyfikatu może wyglądać następująco:

```
# rozszerzenia wymagane do wygenerowania certyfikatu
# self signed przy użyciu openssl

# nie zmieniać tej sekcji
# =======================
[req]
default_bits = 2048
distinguished_name = req_distinguished_name
req_extensions = req_ext
x509_extensions = v3_req
prompt = no
```

```
# wpisać własne wartości w tej sekcji
# ==================================
[req_distinguished_name]
countryName = PL
stateOrProvinceName = RZE
localityName = RZE
organizationName = ACME-INC
commonName = acmeinc.internal
emailAddress = admin@acmeinc.internal

# nie zmieniać tej sekcji
# ========================
[req_ext]
basicConstraints = CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always
subjectAltName = @alt_names

# nie zmieniać tej sekcji
# ========================
[v3_req]
basicConstraints = CA:FALSE
subjectAltName = @alt_names
keyUsage = nonRepudiation,digitalSignature,keyEncipherment
extendedKeyUsage = serverAuth

# domeny trybu development
# ========================
[alt_names]
DNS.1 = localhost
DNS.2 = auth
DNS.3 = api
DNS.4 = tenant-api
DNS.5 = syncfusion-api
DNS.6 = public-api
DNS.7 = mobile-api
```

```
DNS.8 = task-service
DNS.9 = telerik-reports
DNS.10 = documentation
```

Wpisy w sekcji *[alt_names]* powyżej są odpowiednie w przypadku wdrożenia na *localhost*, bez serwera proxy i bez obsługi wielu tenantów. W przypadku uruchomienia domenowego z obsługą wielu tenantów sekcja *[alt_names]* powinna wyglądać następująco:

```
DNS.1 = nax-deploy.local
DNS.2 = *.nax-deploy.local
DNS.3 = auth
DNS.4 = *.auth
DNS.5 = api
DNS.6 = tenant-api
DNS.7 = syncfusion-api
DNS.8 = public-api
DNS.9 = mobile-api
DNS.10 = task-service
DNS.11 = telerik-reports
DNS.12 = documentation
```

Natomiast w przypadku profilu wdrożenia *PROXY* w certyfikacie do komunikacji wewnętrznej nie są potrzebne wpisy *DNS.1* i *DNS.2* z przykładu powyżej.
Plik konfiguracji certyfikatu należy zapisać z rozszerzeniem *cnf*.
Certyfikat generuje się następującym poleceniem:

```
openssl req \
    -x509 \
    -newkey rsa:2048 \
    -nodes \
    -sha256 \
    -days 365 \
    -config path/filename.cnf \
    -keyout d4r-internal.key \
    -out d4r-internal.crt
```

path/filename.cnf to ścieżka i nazwa pliku konfiguracyjnego. Wynikowe pliki muszą się znaleźć w podkatalogu *certs* katalogu głównego wdrożenia, a ich nazwa musi być zgodna z nazwą podaną w pliku *docker.env* dla zmiennej *NAX_ROOT_CERTIFICATE*.

2.3.3. Certyfikat proxy

Dla wdrożenia w trybie proxy potrzebny jest jeszcze certyfikat dla serwera. Jeśli to będzie certyfikat wystawiony przez zaufany urząd certyfikacji, wystarczy go skopiować do podkatalogu *certs*.
Aby natomiast użyć certyfikatu samopodpisanego, należy go wygenerować podobnie jak w przypadku certyfikatu do komunikacji wewnętrznej, zainstalować w magazynie certyfikatów oraz skopiować do podkatalogu *certs* w katalogu głównym wdrożenia.
W tym wypadku plik konfiguracyjny różni się od poprzedniego tylko zawartością ostatniej sekcji, która wygląda następująco:

```
[alt_names]
DNS.1 = *.nax-deploy.local
DNS.2 = nax-deploy.local
```

W poleceniu generowania certyfikatu należy podać prawidłową nazwę pliku konfiguracyjnego (i ścieżkę) oraz nazwę plików wynikowych zgodną ze zmienną *NAX_NGINX_CERTIFICATE* w pliku *docker.env*, na przykład:

```
openssl req \
    -x509 \
    -newkey rsa:2048 \
    -nodes \
    -sha256 \
    -days 365 \
    -config path/filename.cnf \
    -keyout d4r-proxy.key \
    -out d4r-proxy.crt
```

2.3.4. Klaster Kubernetes

Certyfikat dla połączeń wewnętrznych między podami w klastrze Kubernetes musi obsługiwać adresację z uwzględnieniem nazw obiektów *Service* używanych w konfiguracji domyślnej (patrz lista nazw we wcześniejszej sekcji) oraz przestrzeni nazw określonej zmienną *IMG_DEPLOY_NAMESPACE* w plikach obiektów *Deployment* (tutaj *naxiom*). W opisywanym przykładzie sekcja *[alt_names]* w pliku konfiguracji tego certyfikatu wygląda następująco:

```
[alt_names]
DNS.1 = tenant-api-service.naxiom
DNS.2 = *.auth-service.naxiom
```

```
DNS.3 = auth-service.naxiom
DNS.4 = api-service.naxiom
DNS.5 = front-service.naxiom
DNS.6 = admin-service.naxiom
DNS.7 = workflow-service.naxiom
DNS.8 = public-api-service.naxiom
DNS.9 = telerik-reports-service.naxiom
DNS.10 = documentation-service.naxiom
DNS.11 = task-service-service.naxiom
DNS.12 = syncfusion-service.naxiom
DNS.13 = tenant-admin-service.naxiom
DNS.14 = mobile-api-service.naxiom
```

Certyfikat generuje się tak samo, jak dla wdrożenia w trybie *proxy*. W tym przypadku jednak konieczne wygenerowanie plików dla obiektów *Secret* typu *tls* używanych do zabezpieczania aplikacji w klastrach Kubernetes. W tym celu należy użyć polecenia:

```
kubectl create secret tls cert-naxiom-internal --cert=./certs/d4r-internal.crt --key=./certs
kubectl create secret tls cert-naxiom-proxy --cert=./certs/d4r-proxy.crt --key=./certs/d4r-p
```

W przypadku korzystania z oprogramowania *Openshift*, zamiast *kubectl* należy użyć programu *oc* (Openshift CLI).

2.4. Parametry połączenia z SQL Server

Parametry połączenia z serwerem bazy danych definiuje się w pliku *appsettings-custom.json* dla serwisu *tenant-api*. Plik należy umieścić w ścieżce */katalog-główny-wdrożenia/secrets/tenant-api/*. W przypadku uruchamiania poleceniem *docker* wdrożono mechanizm plików chronionych.
W klastrze Kubernetes można użyć tego mechanizmu w momencie definiowania obiektu *Pod*. Wymaga to przekazania odpowiednio argumentów do kontenera, na przykład:

```
args:
  - "/root/.aspnet/configs/appsettings-custom.json:/home/app/appsettings-custom.json")
```

Dodatkowo zaleca się dystrybuowanie plików z danymi poufnymi w obiektach *Secrets*.
Przykładowy plik może wyglądać następująco:

```
{
  "ConnectionStrings": {
```

```
        "SQLConnectionString": "
           Server=host.docker.internal;
           Initial Catalog=tenants-docker;
           User ID=sa;
           Password=p@sswOrd;
           Connection Timeout=30;"
  },
  "AppConfiguration": {
    "FileStorageRoot": "/home/app/nAxiom"
  }
}
```

Parametr *Server* musi mieć określoną wartość, zależnie od używanej instancji serwera:

- *host.docker.internal*: SQL Server zainstalowany na maszynie lokalnej, zarządzanie kontenerami za pomocą programu *docker*;
- *host.minikube.internal*: SQL Server zainstalowany na maszynie lokalnej, zarządzanie kontenerami za pomocą programu *minikube* (lokalny, testowy klaster Kubernetes);
- *mssql*: SQL Server działający jako kontener;

Parametr *Initial Catalog* to nazwa bazy danych z informacjami o tenantach. Po uruchomieniu kontenerów nAxiom, jako pierwszy musi zalogować się administrator tenantów i utworzyć pierwszego tenanta. Ten proces można wykonać automatycznie, używając folderu *toMigrate* w katalogu głównym wdrożenia. Odpowiednią procedurę opisano w drugiej części tej dokumentacji.
We wszystkich przypadkach zakłada się, że serwer bazy danych używa portu domyślnego 1443. W przeciwnym razie należy podać numer portu w parametrach połączenia.
Aby połączyć się z serwerem bazy danych uruchamianym w kontenerze z programu MS SQL Management Studio, jako nazwę serwera należy podać adres IP (lub nazwę hosta) i po przecinku numer portu, np. *localhost, 1443*.

2.5. Pliki konfiguracyjne serwera nginx

Poprawne uruchomienie środowiska nAxiom w trybie proxy wymaga przygotowania przedstawionych poniżej plików konfiguracyjnych. Pliki powinny znaleźć się w ścieżce zgodnej ze wskazaną w pliku *docker-compose-extra.yml*.

> Uwaga: ze względu na dłuższy czas przetwarzania rozbudowanych żądań zaleca się wydłużenie limitu czas dla operacji odczytu, nawiązywania połączenia i wysyłania z

> domyślnych 60 s na 300 s. W tym celu w konfiguracji serwera nginx należy dodać następujące wpisy:
>
> ```
> proxy_read_timeout 300s;
> proxy_connect_timeout 300s;
> proxy_send_timeout 300s;
> ```

2.5.1. Plik naxiom.conf

```
server {
    # Nasłuchiwanie połączeń SSL na porcie 443
    listen 443 ssl;
    # Ustawienie domeny ${DOMAIN_NAME}, można ustawić kilka domen.
    server_name ${DOMAIN_NAME} *.${DOMAIN_NAME};

    # Konfiguracja certyfikatu w pliku pomocniczym
    include helpers/naxiom-certificate.conf;
    # Konfiguracja buforów (cookie, nagłówki, zapytania itp.).
    include helpers/naxiom-buffers.conf;

    # Konfiguracja limitów czasu.
    resolver 127.0.0.11 valid=30s;

    proxy_read_timeout 300s;
    proxy_connect_timeout 300s;
    proxy_send_timeout 300s;

    # AUTH
    location / {
        proxy_pass https://auth:5001;
        # Konfiguracja nagłówków dla żądań używających proxy_pass;
        # konfigurację można zmienić, edytując plik pomocniczy.
        include helpers/naxiom-headers.conf;
    }

    # API
```

```
    location /back/ {
        proxy_pass https://api:5103/;
        include helpers/naxiom-headers.conf;
    }

    # Documentation
    location /docapi/ {
        proxy_pass https://documentation:5401/;
        include helpers/naxiom-headers.conf;
    }

    # TelerikReports
    location /reportsapi/ {
        proxy_pass https://telerik-reports:5503/;
        include helpers/naxiom-headers.conf;
    }

    # MobileAPI
    location /mobileapi/ {
        proxy_pass https://mobile-api:5503/;
        include helpers/naxiom-headers.conf;
    }

    # PublicAPI
    location /publicapi/ {
        proxy_pass https://public-api:5203/;
        include helpers/naxiom-headers.conf;
    }

    # TenantsAPI
    location /tenantsapi/ {
        proxy_pass https://tenant-api:5803/;
        include helpers/naxiom-headers.conf;
    }

    # FrontSPA
    location /front/ {
```

```
        proxy_pass https://front:4200;
        include helpers/naxiom-headers.conf;
    }

    # AdminSPA
    location /admin/ {
        proxy_pass https://admin:4300;
        include helpers/naxiom-headers.conf;
    }

    # WorkflowSPA
    location /workflow/ {
        proxy_pass https://workflow:4400;
        include helpers/naxiom-headers.conf;
    }

    # TenantsAdminSPA
    location /tenantsadmin/ {
        proxy_pass https://tenant-admin:4800;
        include helpers/naxiom-headers.conf;
    }
}
```

2.5.2. Plik naxiom-certificate.conf

```
# Set certificate for HTTPS
ssl_certificate            cert/proxy.crt;
ssl_certificate_key        cert/proxy.key;

# Set other certificate options
ssl_session_cache           shared:SSL:1m;
ssl_session_timeout         5m;
ssl_ciphers                 HIGH:!aNULL:!MD5;
ssl_prefer_server_ciphers   on;
```

2.5.3. Plik naxiom-buffers.conf

```
proxy_busy_buffers_size     512k;
proxy_buffers               4 512k;
proxy_buffer_size           256k;
large_client_header_buffers 4 32k;
client_max_body_size        100m;
```

2.5.4. Plik naxiom-headers.conf

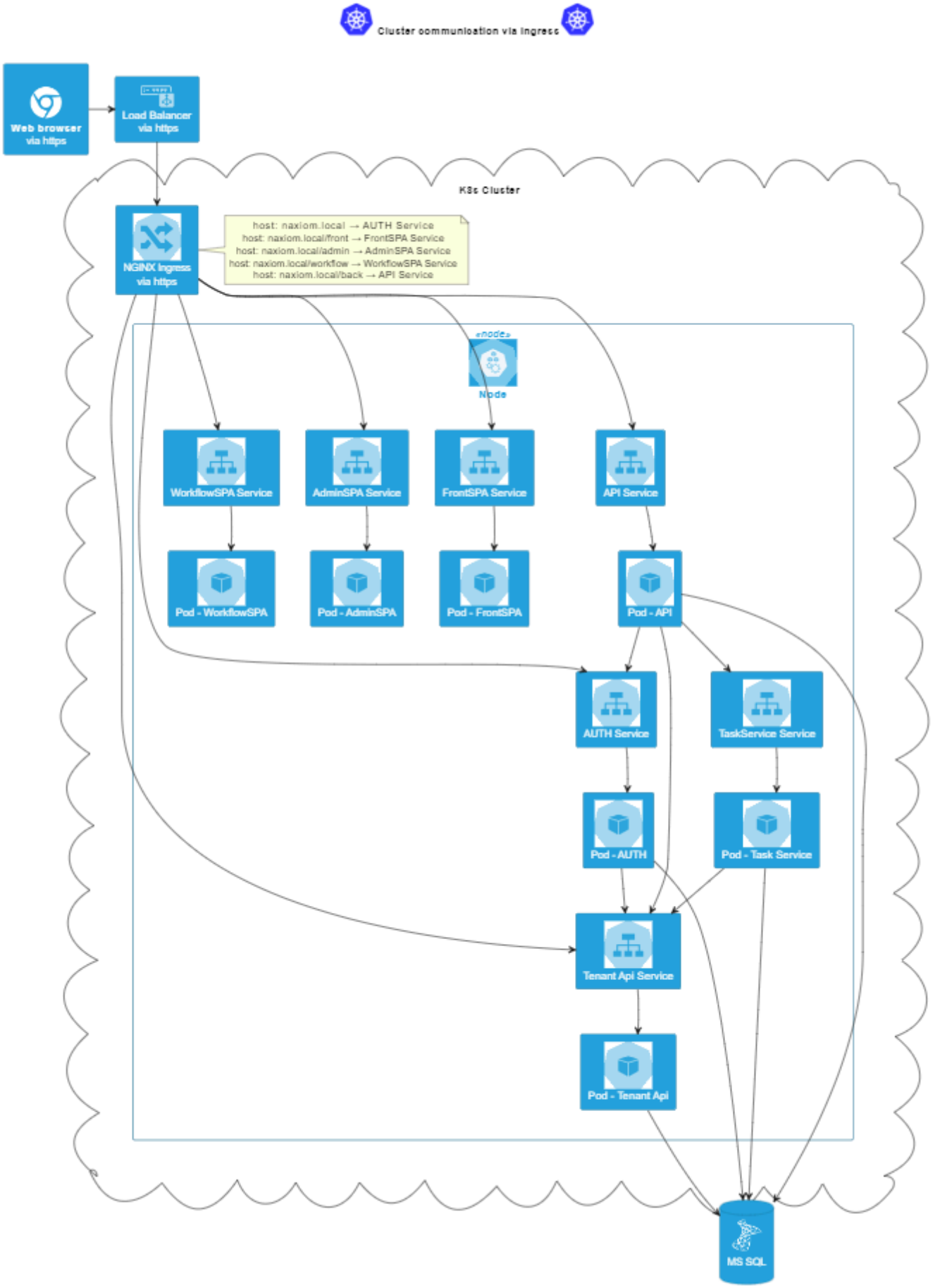
```
proxy_set_header Host              $host;
proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Host  $host:$server_port;
proxy_set_header X-Real-IP         $remote_addr;
```

2.6. Konfiguracja serwera proxy dla wdrożenia w klastrze

Do mapowania ruchu przychodzącego na pody indywidualnych serwisów nAxiom używane są obiekty *Ingress* w Kubernetes lub *Route* w Openshift.

2.6.1. Ingress/Kubernetes

Żądania kierowane do klastra kontenerów Kubernetes są odbierane przez obiekt *Ingress*, który używa serwera nginx, i kierowane do odpowiednich podów, za pośrednictwem obsługujących je obiektów *Service*, jak na poniższej ilustracji.


Cluster communication via Ingress
Web browser via https
Load Balancer via https
K8s Cluster
NGINX Ingress via https
host: naxiom.local → AUTH Service
host: naxiom.local/front → FrontSPA Service
host: naxiom.local/admin → AdminSPA Service
host: naxiom.local/workflow → WorkflowSPA Service
host: naxiom.local/back → API Service
«node»
Node
WorkflowSPA Service
AdminSPA Service
FrontSPA Service
API Service
Pod - WorkflowSPA
Pod - AdminSPA
Pod - FrontSPA
Pod - API
AUTH Service
TaskService Service
Pod - AUTH
Pod - Task Service
Tenant Api Service
Pod - Tenant Api
MS SQL

Obiekty *Ingress* definiuje się osobno dla serwisów dotnetowych i angularowych, a także dla serwisu *auth*. Definicje obiektów zawierają reguły odpowiadające za prawidłowe kierowanie ruchu.
Poniżej zamieszczono zawartość trzech plików konfiguracyjnych.
Konfiguracja dla serwisów dotnetowych (plik *naxiom-ingress-back.yaml*):

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: naxiom-ingress-back
  namespace: {{ .Values.namespace }}
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: 'HTTPS'
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/proxy-buffering: "on"
    nginx.ingress.kubernetes.io/proxy-buffer-size: "128k"
    nginx.ingress.kubernetes.io/proxy-buffers-number: "4"
    nginx.ingress.kubernetes.io/proxy-body-size: 64m
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - '{{ .Values.ingress.dnsName }}'
        - '*.{{ .Values.ingress.dnsName }}'
      secretName: cert-naxiom
  rules:
    - host: '{{ .Values.ingress.dnsName }}'
      http:
        paths:
          - path: /back(/|$)(.*)
            pathType: Prefix
            backend:
              service:
                name: api-service
                port:
```

```
                number: 5103
        - path: /docapi(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: documentation-service
              port:
                number: 5401
        - path: /publicapi(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: public-api-service
              port:
                number: 5203
        - path: /mobileapi(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: mobile-api-service
              port:
                number: 5503
        - path: /tenantsapi(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: tenant-api-service
              port:
                number: 5803
        - path: /reportsapi(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: telerik-reports-service
              port:
                number: 5503
  - host: '*.{{ .Values.ingress.dnsName }}'
```

```
    http:
      paths:
        - path: /back(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: api-service
              port:
                number: 5103
        - path: /docapi(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: documentation-service
              port:
                number: 5401
        - path: /publicapi(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: public-api-service
              port:
                number: 5203
        - path: /mobileapi(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: mobile-api-service
              port:
                number: 5503
        - path: /reportsapi(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: telerik-reports-service
              port:
                number: 5503
```

Konfiguracja dla serwisów angularowych (plik *naxiom-ingress-front.yaml*):

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: naxiom-ingress-front
  namespace: {{ .Values.namespace }}
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: 'HTTPS'
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/rewrite-target: /$1/$3
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - '{{ .Values.ingress.dnsName }}'
        - '*.{{ .Values.ingress.dnsName }}'
      secretName: cert-naxiom
  rules:
    - host: '{{ .Values.ingress.dnsName }}'
      http:
        paths:
          - path: /(front)(/|$)(.*)
            pathType: Prefix
            backend:
              service:
                name: front-service
                port:
                  number: 4200
          - path: /(admin)(/|$)(.*)
            pathType: Prefix
            backend:
              service:
                name: admin-service
                port:
                  number: 4300
          - path: /(workflow)(/|$)(.*)
```

```
          pathType: Prefix
          backend:
            service:
              name: workflow-service
              port:
                number: 4400
        - path: /(tenantsadmin)(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: tenant-admin-service
              port:
                number: 4800
  - host: '*.{{ .Values.ingress.dnsName }}'
    http:
      paths:
        - path: /(front)(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: front-service
              port:
                number: 4200
        - path: /(admin)(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: admin-service
              port:
                number: 4300
        - path: /(workflow)(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: workflow-service
              port:
                number: 4400
```

```
          - path: /(tenantsadmin)(/|$)(.*)
            pathType: Prefix
            backend:
              service:
                name: tenant-admin-service
                port:
                  number: 4800
```

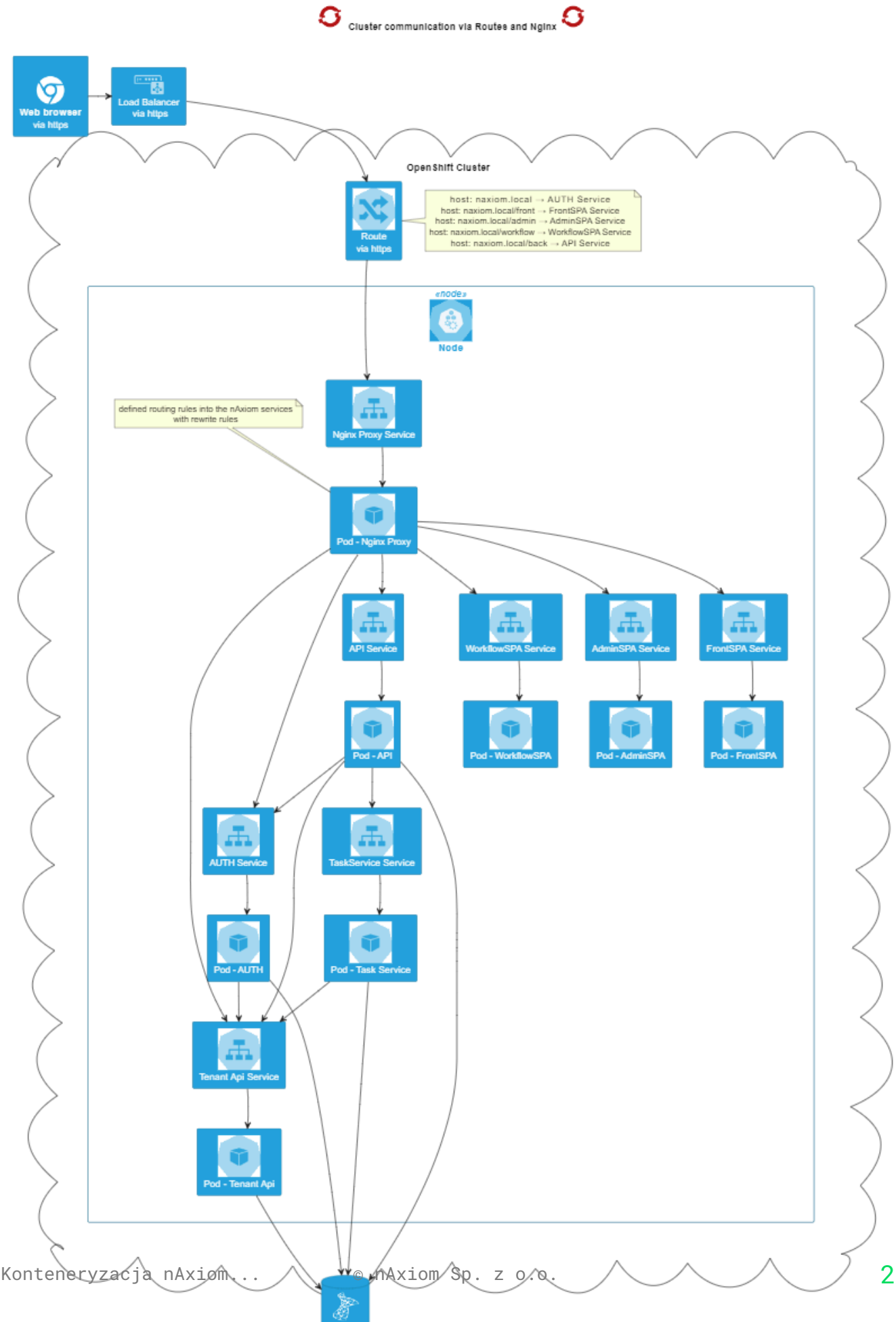
Konfiguracja dla serwisu *auth* (plik *naxiom-ingress.yaml*):

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: naxiom-ingress
  namespace: {{ .Values.namespace }}
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: 'HTTPS'
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - '{{ .Values.ingress.dnsName }}'
        - '*.{{ .Values.ingress.dnsName }}'
      secretName: cert-naxiom
  rules:
    - host: '{{ .Values.ingress.dnsName }}'
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: auth-service
                port:
                  number: 5001
    - host: '*.{{ .Values.ingress.dnsName }}'
```

```
    http:
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: auth-service
              port:
                number: 5001
```

2.6.2. Route/Openshift

W przypadku oprogramowania Openshift obiekt *Route* znajduje się na zewnątrz klastra i kieruje żądania do poda nginx, który działa w węźle i rozdziela ruch na pody nAxiom na podstawie konfiguracji zdefiniowanej w obiektach *config map* (takiej jak opisana w sekcji Pliki konfiguracyjne serwera nginx). Architekturę przedstawiono na ilustracji.


Cluster communication via Routes and Nginx
Web browser via https
Load Balancer via https
OpenShift Cluster
Route via https
host: naxiom.local → AUTH Service
host: naxiom.local/front → FrontSPA Service
host: naxiom.local/admin → AdminSPA Service
host: naxiom.local/workflow → WorkflowSPA Service
host: naxiom.local/back → API Service
«node»
Node
defined routing rules into the nAxiom services with rewrite rules
Nginx Proxy Service
Pod - Nginx Proxy
API Service
WorkflowSPA Service
AdminSPA Service
FrontSPA Service
Pod - API
Pod - WorkflowSPA
Pod - AdminSPA
Pod - FrontSPA
AUTH Service
TaskService Service
Pod - AUTH
Pod - Task Service
Tenant Api Service
Pod - Tenant Api

Obiekt *Route* używa dwóch plików konfiguracyjnych.
Plik *naxdev-local-route.yaml*:

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: naxdev-local-route
  namespace: naxiom
spec:
  host: naxdev.local
  to:
    kind: Service
    name: nginx-service
    weight: 100
  port:
    targetPort: 443
  tls:
    termination: passthrough
    insecureEdgeTerminationPolicy: Redirect
  wildcardPolicy: None
```

Plik *wildcard-naxdev-local-route.yaml*:

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: wildcard-naxdev-local-route
  namespace: naxiom
spec:
  host: wildcard.naxdev.local
  to:
    kind: Service
    name: nginx-service
    weight: 100
  port:
    targetPort: 443
```

```
  tls:
    termination: passthrough
    insecureEdgeTerminationPolicy: Redirect
  wildcardPolicy: Subdomain
```

2.7. Ładowanie obrazów do repozytorium

Zestaw obrazów nAxiom jest dystrybuowany w formie archiwum *tar*. Aby załadować obrazy do lokalnego repozytorium obrazów, użyj polecenia:

```
docker load -i naxiom-images.tar
```

Sprawdź, czy obrazy zostały załadowane, używając polecenia

```
docker image ls -a
```

2.8. Pliki docker compose

Do uruchomienia środowiska nAxiom z zestawu obrazów, najwygodniej użyć pliku *docker-compose.yml* z ustawieniami konfiguracyjnymi dla poszczególnych serwisów. Ze względu na dużą liczbę obrazów, przygotowano kilka takich plików, które zostaną opisane w kolejnych sekcjach. Zakłada się, że wszystkie pliki *compose* znajdują się w katalogu głównym wdrożenia, podobnie jak plik ustawień środowiskowych *docker.env*.

2.8.1. docker-compose.yml

Poniżej przedstawiono plik *docker-compose* dla serwisów nAxiom. Plik zawiera podstawowe parametry, takie jak nazwa i tag obrazu, zależności, profile uruchamiania oraz nazwę domeny i typ wdrożenia. Użycie polecenia `docker compose --profile` pozwala uruchomić środowisko nAxiom z wszystkimi lub z wybranymi serwisami. Zestaw minimalny obejmuje serwisy: *auth*, *api*, *tenant-api* i *front*.

```
version: '3.7'

services:
# nazwa serwisu
  auth:
# nazwa i tag obrazu (z pliku docker.env)
    image: ${DOCKER_REGISTRY-}auth:${NAX_VERSION}
# zależności
```

```
    depends_on:
      - api
# profile uruchamiania
    profiles:
      - full
      - medium
      - minimal
# nazwa domeny i typ wdrożenia (z pliku docker.env)
    environment:
      IMG_DOMAIN_NAME: ${DOMAIN_NAME}
      IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
  api:
    image: ${DOCKER_REGISTRY-}api:${NAX_VERSION}
    depends_on:
      - auth
    profiles:
      - full
      - medium
      - minimal
    environment:
      IMG_DOMAIN_NAME: ${DOMAIN_NAME}
      IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
    # test stanu
    healthcheck:
      test: ['CMD', 'curl', '--insecure', '--fail', 'https://api:5103/isAlive/Ping']
      interval: 10s
      timeout: 5s
      retries: 30
  tenant-api:
    image: ${DOCKER_REGISTRY-}tenant-api:${NAX_VERSION}
    profiles:
      - full
      - medium
      - minimal
    environment:
      IMG_DOMAIN_NAME: ${DOMAIN_NAME}
      IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
```

```
    healthcheck:
      test: ['CMD', 'curl', '--insecure', '--fail', 'https://tenant-api:5803/isAlive']
      interval: 10s
      timeout: 5s
      retries: 20
  syncfusion-api:
    image: ${DOCKER_REGISTRY-}syncfusion-api:${NAX_VERSION}
    profiles:
      - full
    environment:
      IMG_DOMAIN_NAME: ${DOMAIN_NAME}
      IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
  # serwis zostanie uruchomiony pod warunkiem
  # poprawnego uruchomienia serwisu w sekcji depends_on
  public-api:
    image: ${DOCKER_REGISTRY-}public-api:${NAX_VERSION}
    depends_on:
      api:
        condition: service_healthy
    profiles:
      - full
    environment:
      IMG_DOMAIN_NAME: ${DOMAIN_NAME}
      IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
  mobile-api:
    image: ${DOCKER_REGISTRY-}mobile-api:${NAX_VERSION}
    depends_on:
      api:
        condition: service_healthy
    profiles:
      - full
    environment:
      IMG_DOMAIN_NAME: ${DOMAIN_NAME}
      IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
  task-service:
    image: ${DOCKER_REGISTRY-}task-service:${NAX_VERSION}
    depends_on:
```

```
    api:
      condition: service_healthy
  profiles:
    - full
    - medium
  environment:
    IMG_DOMAIN_NAME: ${DOMAIN_NAME}
    IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
telerik-reports:
  image: ${DOCKER_REGISTRY-}telerik-reports:${NAX_VERSION}
  depends_on:
    api:
      condition: service_healthy
  profiles:
    - full
  environment:
    IMG_DOMAIN_NAME: ${DOMAIN_NAME}
    IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
documentation:
  image: ${DOCKER_REGISTRY-}documentation:${NAX_VERSION}
  depends_on:
    api:
      condition: service_healthy
  profiles:
    - full
    - medium
  environment:
    IMG_DOMAIN_NAME: ${DOMAIN_NAME}
    IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
front:
  image: ${DOCKER_REGISTRY-}front:${NAX_VERSION}
  depends_on:
    - api
  profiles:
    - full
    - medium
    - minimal
```

```
  environment:
    IMG_DOMAIN_NAME: ${DOMAIN_NAME}
    IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
admin:
  image: ${DOCKER_REGISTRY-}admin:${NAX_VERSION}
  depends_on:
    - api
  profiles:
    - full
    - medium
  environment:
    IMG_DOMAIN_NAME: ${DOMAIN_NAME}
    IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
workflow:
  image: ${DOCKER_REGISTRY-}workflow:${NAX_VERSION}
  depends_on:
    - admin
  profiles:
    - full
    - medium
  environment:
    IMG_DOMAIN_NAME: ${DOMAIN_NAME}
    IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
tenant-admin:
  image: ${DOCKER_REGISTRY-}tenant-admin:${NAX_VERSION}
  depends_on:
    - tenant-api
  profiles:
    - full
    - medium
  environment:
    IMG_DOMAIN_NAME: ${DOMAIN_NAME}
    IMG_DEPLOY_TYPE: ${DEPLOY_TYPE}
```

2.8.2. docker-compose-common.yml

Ten plik zawiera informacje o mapowaniu portów używanych przez serwisy w kontenerze i na zewnątrz. Wymagane jest pozostawienie tych ustawień bez zmian. Ponadto, w kluczu *command* można wskazać pliki ustawień, które mają być traktowane w kontenerze jako chronione. Taki plik zostanie wczytany do kontenera i usunięty. Plik w oryginalnej lokalizacji nie będzie dostępny z poziomu kontenera. Patrz także Usuwanie plików ustawień.

```
version: '3.7'

services:
  # nazwa serwisu nAxiom
  auth:
    # mapowanie portów serwisu, pierwszy to port
    # używany na zewnątrz, a drugi to port używany wewnętrznie w kontenerze
    # (tego portu nie można zmienić); aby serwis był dostępny na porcie np. 6010,
    # należy zmienić wartość na 6010:5001;
    # ta zmiana wymaga odpowiedniej zmiany konfiguracji
    ports:
      - 5001:5001
  api:
    # pozycja command służy do wskazywania plików "chronionych"
    # takie pliki zostaną wczytane do kontenera podczas uruchamiania,
    # a następnie usunięte (w kontenerze, nie na hoście);
    # pierwsza ścieżka to miejsce docelowe w kontenerze,
    # druga to położenie źródłowe pliku w katalogu głównym wdrożenia
    command: '/root/.aspnet/configs/appsettings-custom.json:
              /home/app/appsettings-custom.json'
    ports:
      - 5103:5103
  tenant-api:
    command: '/root/.aspnet/configs/appsettings-custom.json:
              /home/app/appsettings-custom.json'
    ports:
      - 5803:5803
  public-api:
    ports:
```

```
      - 5203:5203
  mobile-api:
    ports:
      - 5503:5503
  telerik-reports:
    # mapowanie jest konieczne, ponieważ serwisy mobile-api i telerik-reports
    # używają tych samych portów w kontenerze
    ports:
      - 5533:5503
  documentation:
    ports:
      - 5401:5401
  front:
    ports:
      - 4200:4200
  admin:
    ports:
      - 4300:4300
  workflow:
    ports:
      - 4400:4400
  tenant-admin:
    ports:
      - 4800:4800
```

2.8.3. docker-compose-volumes.yml

W tym pliku deklaruje się mapowanie ścieżek między kontenerem i hostem. Dotyczy to własnych plików konfiguracyjnych, certyfikatów, plików skryptów oraz katalogów w systemie plików hosta na potrzeby zapisu danych z kontenera.

```
version: '3.7'

services:
  auth:
    volumes:
# własny plik konfiguracyjny logowania informacji dla serwisu auth
```

```
      - '${NAX_ROOT_CONF_DIR}/configs/auth/nlog-custom.config:
        /home/app/nlog-custom.config:ro'
# własny plik konfiguracyjny aplikacji (serwisu) auth
      - '${NAX_ROOT_CONF_DIR}/configs/auth/appsettings-custom.json:
        /home/app/appsettings-custom.json:ro'
# plik keytab na potrzeby logowania z uwierzytelnianiem Windows
      - '${NAX_ROOT_CONF_DIR}/secrets/auth/krb5.keytab:
        /etc/krb5.keytab:ro'
# pliki certyfikatu
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
        /usr/local/share/ca-certificates/certificate.crt:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
        /root/.aspnet/https/certificate.key:ro'
  api:
    volumes:
      - '${NAX_ROOT_CONF_DIR}/configs/api/nlog-custom.config:
        /home/app/nlog-custom.config:ro'
      - '${NAX_ROOT_CONF_DIR}/secrets/api/appsettings-custom.json:
        /root/.aspnet/configs/appsettings-custom.json:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
        /usr/local/share/ca-certificates/certificate.crt:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
        /root/.aspnet/https/certificate.key:ro'
  tenant-api:
    volumes:
      - '${NAX_ROOT_CONF_DIR}/configs/tenant-api/nlog-custom.config:
        /home/app/nlog-custom.config:ro'
      - '${NAX_ROOT_CONF_DIR}/secrets/tenant-api/appsettings-custom.json:
        /root/.aspnet/configs/appsettings-custom.json:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
        /usr/local/share/ca-certificates/certificate.crt:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
        /root/.aspnet/https/certificate.key:ro'
# folder z plikami konfiguracyjnymi appsettings.json
# używany do symulacji aktualizacji istniejącego środowiska nAxiom
# oraz utworzenia pierwszego tenanta
      - '${NAX_ROOT_CONF_DIR}/toMigrate/:/home/app/toMigrate:ro'
```

```
  syncfusion-api:
    volumes:
      - '${NAX_ROOT_CONF_DIR}/configs/syncfusion-api/nlog-custom.config:
        /home/app/nlog-custom.config:ro'
      - '${NAX_ROOT_CONF_DIR}/configs/syncfusion-api/appsettings-custom.json:
        /home/app/appsettings-custom.json:ro'
# własny skrypt z poleceniami wykonywanymi podczas uruchamiania serwisu
      - '${NAX_ROOT_CONF_DIR}/scripts/syncfusion-api/custom-entrypoint.sh:
        /home/app/custom-entrypoint.sh:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
        /usr/local/share/ca-certificates/certificate.crt:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
        /root/.aspnet/https/certificate.key:ro'
  public-api:
    volumes:
      - '${NAX_ROOT_CONF_DIR}/configs/public-api/nlog-custom.config:
        /home/app/nlog-custom.config:ro'
      - '${NAX_ROOT_CONF_DIR}/configs/public-api/appsettings-custom.json:
        /home/app/appsettings-custom.json:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
        /usr/local/share/ca-certificates/certificate.crt:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
        /root/.aspnet/https/certificate.key:ro'
  mobile-api:
    volumes:
      - '${NAX_ROOT_CONF_DIR}/configs/mobile-api/nlog-custom.config:
        /home/app/nlog-custom.config:ro'
      - '${NAX_ROOT_CONF_DIR}/configs/mobile-api/appsettings-custom.json:
        /home/app/appsettings-custom.json:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
        /usr/local/share/ca-certificates/certificate.crt:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
        /root/.aspnet/https/certificate.key:ro'
  task-service:
    volumes:
      - '${NAX_ROOT_CONF_DIR}/configs/task-service/nlog-custom.config:
        /home/app/nlog-custom.config:ro'
```

```
    - '${NAX_ROOT_CONF_DIR}/configs/task-service/appsettings-custom.json:
      /home/app/appsettings-custom.json:ro'
    - '${NAX_ROOT_CONF_DIR}/scripts/task-service/custom-entrypoint.sh:
      /home/app/custom-entrypoint.sh:ro'
    - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
      /usr/local/share/ca-certificates/certificate.crt:ro'
    - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
      /root/.aspnet/https/certificate.key:ro'
telerik-reports:
  volumes:
    - '${NAX_ROOT_CONF_DIR}/configs/telerik-reports/nlog-custom.config:
      /home/app/nlog-custom.config:ro'
    - '${NAX_ROOT_CONF_DIR}/configs/telerik-reports/appsettings-custom.json:
      /home/app/appsettings-custom.json:ro'
    - '${NAX_ROOT_CONF_DIR}/scripts/telerik-reports/custom-entrypoint.sh:
      /home/app/custom-entrypoint.sh:ro'
    - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
      /usr/local/share/ca-certificates/certificate.crt:ro'
    - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
      /root/.aspnet/https/certificate.key:ro'
documentation:
  volumes:
    - '${NAX_ROOT_CONF_DIR}/configs/documentation/nlog-custom.config:
      /home/app/nlog-custom.config:ro'
    - '${NAX_ROOT_CONF_DIR}/configs/documentation/appsettings-custom.json:
      /home/app/appsettings-custom.json:ro'
    - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
      /usr/local/share/ca-certificates/certificate.crt:ro'
    - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
      /root/.aspnet/https/certificate.key:ro'
front:
  volumes:
    - '${NAX_ROOT_CONF_DIR}/configs/front/config-custom.json:
      /html/front/assets/config/config-custom.json:ro'
    - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
      /usr/local/share/ca-certificates/certificate.crt:ro'
    - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
```

```
        /root/.node/https/certificate.key:ro'
  admin:
    volumes:
      - '${NAX_ROOT_CONF_DIR}/configs/admin/config-custom.json:
        /html/admin/assets/config/config-custom.json:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
        /usr/local/share/ca-certificates/certificate.crt:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
        /root/.node/https/certificate.key:ro'
  workflow:
    volumes:
      - '${NAX_ROOT_CONF_DIR}/configs/workflow/config-custom.json:
        /html/workflow/assets/config/config-custom.json:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
        /usr/local/share/ca-certificates/certificate.crt:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
        /root/.node/https/certificate.key:ro'
  tenant-admin:
    volumes:
      - '${NAX_ROOT_CONF_DIR}/configs/tenant-admin/config-custom.json:
        /html/tenantsadmin/assets/config/config-custom.json:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.crt:
        /usr/local/share/ca-certificates/certificate.crt:ro'
      - '${NAX_ROOT_CONF_DIR}/certs/${NAX_ROOT_CERTIFICATE}.key:
        /root/.node/https/certificate.key:ro'
```

2.8.4. docker-compose-extra.yml

Ten plik zawiera konfigurację uruchamiania dodatkowych kontenerów: MS SQL Server oraz nginx. Są one uruchamiane w przypadku użycia w poleceniu `docker compose` opcji `--profiles` o wartości odpowiednio *db* i *proxy*. Odpowiednie pliki konfiguracyjne muszą znajdować się w podkatalogach *mssql* i *nginx* w katalogu głównym wdrożenia.

```
services:
# zależność serwisu tenant-api uruchamianego w profilu db
# od poprawnego uruchomienia serwisu mssql
  tenant-api:
```

```
  profiles:
    - db
  depends_on:
    mssql:
      condition: service_healthy
# konfiguracja dodatkowych serwisów, które będą uruchamiane jako kontenery
mssql:
  image: mcr.microsoft.com/mssql/server:${NAX_MSSQL_VERSION}
  # mapowanie portów 1433 na 1443 zapewni dostęp w środowisku lokalnym;
  # można usunąć tę opcję w środowisku produkcyjnym
  ports:
    - 1443:1433
  # dostosowanie kontenera za pomocą zmiennych środowiskowych;
  # więcej informacji znajduje się na stronie mssql w docker hub
  environment:
    ACCEPT_EULA: Y
    MSSQL_SA_PASSWORD: ${NAX_MSSQL_PASSWORD}
    MSSQL_COLLATION: 'Polish_CI_AS'
  healthcheck:
    test: ['CMD', '/opt/mssql-tools/bin/sqlcmd', '-U', 'sa', '-P',
           '${NAX_MSSQL_PASSWORD}', '-Q', 'SELECT 1']
    interval: 10s
    timeout: 5s
    retries: 5
  # wolumen do zapisu bazy danych w systemie plików hosta;
  # jeśli nie zostanie zdefiniowany, każde zatrzymanie kontenera
  # spowoduje utratę danych
  volumes:
    - '${NAX_MSSQL_DATA_DIR}:/var/opt/mssql/data'
  # profil, którego należy użyć, aby uruchomić kontener SQL Server
  profiles:
    - db

# W systemie MacOS z procesorem ARM można użyć poniższych ustawień
# (niezalecane w środowiskach produkcyjnych);
# wymaga zmiany parametru NAX_MSSQL_VERSION (docker.env) na "latest",
# ponieważ obraz Azure SQL Edge nie ma przedrostku roku
```

```
# mssql:
#   image: mcr.microsoft.com/azure-sql-edge:${NAX_MSSQL_VERSION}
#   ports:
#     - 1443:1433
#   environment:
#     ACCEPT_EULA: 1
#     MSSQL_SA_PASSWORD: ${NAX_MSSQL_PASSWORD}
#   volumes:
#     - '${NAXIOM_MSSQL_DATA_DIRECTORY}:/var/opt/mssql/data'
#   profiles:
#     - db

# Wdrożenie w trybie proxy wymaga użycia serwera nginx;
# aby korzystać z instancji nginx zainstalowanej na hoście, wykomentuj tę sekcję
nginx:
  image: nginx:${NAX_NGINX_VERSION}
  command: >
    /bin/bash -c "([[ -f /etc/nginx/naxiom.template.conf ] && envsubst <
    /etc/nginx/naxiom.template.conf > /etc/nginx/conf.d/default.conf)  ||
    (:)) && exec nginx -g 'daemon off;'"
  # na hoście używany jest tylko port 443; nAxiom nie korzysta z portu 80
  ports:
    - 443:443
  environment:
    DOMAIN_NAME: ${DOMAIN_NAME}
  volumes:
    # certyfikaty (crt i key) dla serwera nginx
    - '${NAX_ROOT_CONF_DIR}/certs/${NAX_NGINX_CERTIFICATE}.crt:
       /etc/nginx/cert/proxy.crt:ro'
    - '${NAX_ROOT_CONF_DIR}/certs/${NAX_NGINX_CERTIFICATE}.key:
       /etc/nginx/cert/proxy.key:ro'
    # główny plik konfiguracyjny z serwisami nAxiom
    - '${NAX_ROOT_CONF_DIR}/nginx/naxiom.conf:
       /etc/nginx/naxiom.template.conf:ro'
    # konfiguracje pomocnicze uwzględniane w pliku default.conf
    # dotyczą buforów zapytań/plików cookie itp., certyfikatów
    # i nagłówków wysyłanych przez proxy_pass
```

```
    - '${NAX_ROOT_CONF_DIR}/nginx/naxiom-buffers.conf:
       /etc/nginx/helpers/naxiom-buffers.conf:ro'
    - '${NAX_ROOT_CONF_DIR}/nginx/naxiom-certificate.conf:
       /etc/nginx/helpers/naxiom-certificate.conf:ro'
    - '${NAX_ROOT_CONF_DIR}/nginx/naxiom-headers.conf:
       /etc/nginx/helpers/naxiom-headers.conf:ro'
  # aby uruchomić kontener serwera nginx, należy uruchomić polecenie
  # docker-compose z dodatkowym profilem proxy
  profiles:
    - proxy
  depends_on:
    - tenant-api
    - auth
    - api
    - public-api
    - mobile-api
    - syncfusion-api
    - telerik-reports
    - tenant-admin
    - workflow
    - front
    - admin
    - task-service
    - documentation
```

2.9. Utworzenie pierwszego tenanta

nAxiom obsługuje tryb *multi-tenancy*, w którym z danego środowiska mogą korzystać różne organizacje, przy czym zarówno konfiguracje aplikacji używanych przez te organizacje, jak i przetwarzane przez nie dane biznesowe są zapisywane w odrębnych bazach danych. Jeśli środowisko nAxiom jest wdrażane przy użyciu instalatora, pierwszy tenant jest definiowany podczas instalacji, a jeśli jest to aktualizacja z wersji wcześniejszej, pierwszy tenant jest tworzony na podstawie wcześniejszej konfiguracji.
W przypadku wdrożenia nAxiom z zestawu obrazów można utworzyć pierwszego tenanta na dwa sposoby:

- uruchomienie kontenerów serwisów *tenant-admin* i *tenant-api* i zdefiniowanie pierwszego tenanta w aplikacji *TenantAdminSPA*.

- przygotowanie zestawu plików konfiguracyjnych z istniejącej instalacji nAxiom (w katalogu *toMigrate*) i automatyczne utworzenie na ich podstawie pierwszego tenanta podczas uruchamiania pełnego zestawu kontenerów.

W przypadku wdrożenia w klastrze Kubernetes zaleca się skorzystanie z drugiej metody i automatycznego utworzenia pierwszego tenanta.

> Info:W razie skorzystania z metody pierwszej, należy wykomentować w pliku *docker-compose-volumes.yml* linię z mapowaniem folderu *ToMigrate* albo utworzyć pusty folder o tej nazwie w głównym katalogu wdrożenia.

2.9.1. Uruchomienie aplikacji TenantAdminSPA

Aplikacja TenantAdminSPA opiera się na serwisie *tenant-admin* i dodatkowo wymaga do działania serwisu *tenant-api*. Aby uruchomić tylko te dwa serwisy, należy w pliku *docker-compose.yml* w sekcji *profiles* tych dwóch serwisów dodać profil *first-tenant*, np.:

```
...

  tenant-api:
    image: ${DOCKER_REGISTRY-}tenant-api:${NAX_VERSION}
    profiles:
      - full
      - medium
      - minimal
      - first-tenant

...
```

Ponadto, aby uruchomić aplikację *TenantAdminSPA* w trybie proxy konieczne jest ograniczenie konfiguracji serwera nginx tylko do dwóch potrzebnych serwisów. W tym celu należy zmodyfikować plik konfiguracyjny *naxiom.conf* (w katalogu *NAX_ROOT_CONF_DIR/nginx*), usuwając z niego (lub komentując) wpisy dotyczące wszystkich serwisów poza *tenant-admin* i *tenant-api*. Po zmianach plik powinien wyglądać następująco:

```
server {
    listen 443 ssl;
    server_name ${DOMAIN_NAME} *.${DOMAIN_NAME};
    include helpers/naxiom-certificate.conf;
```

```
    include helpers/naxiom-buffers.conf;
    location /tenantsapi/ {
        proxy_pass https://tenant-api:5803/;
        include helpers/naxiom-headers.conf;
    }
    location /tenantsadmin/ {
        proxy_pass https://tenant-admin:4800;
        include helpers/naxiom-headers.conf;
    }
}
```

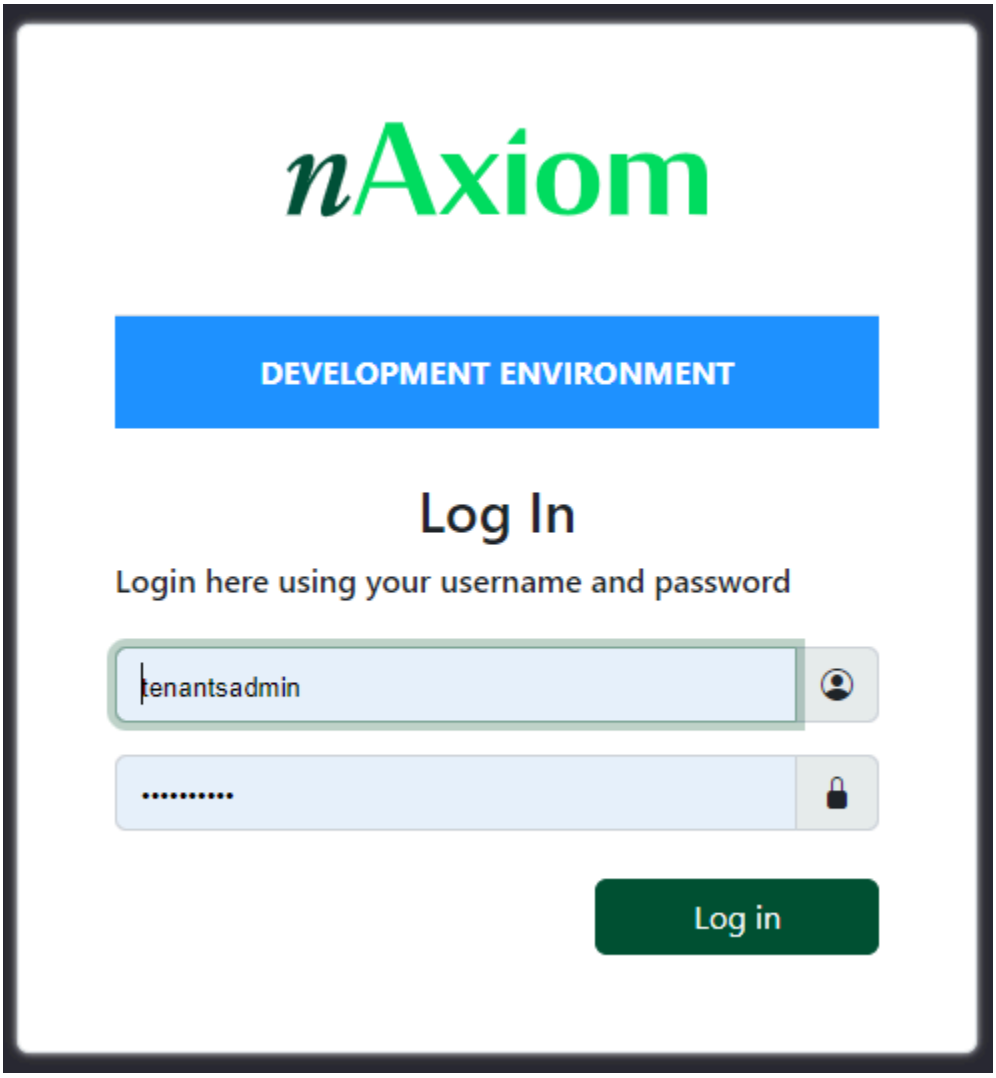
Jeżeli plik zostanie zapisany pod zmienioną nazwą, należy także zaktualizować tę nazwę w pliku *docker-compose-extra.yml*.
Po dokonaniu powyższych zmian należy użyć polecenia (używana wersja polecenia *docker compose* to v2):

```
docker compose \
    -f docker-compose.yml \
    -f docker-compose-common.yml \
    -f docker-compose-extra.yml \
    -f docker-compose-volumes.yml \
    --env-file docker.env \
    --profile first-tenant \
    --profile db \
    --profile proxy \
    up -d
```

> Uwaga: Opcja *–profile db* jest wymagana tylko wtedy, jeśli ma być używana instancja bazy danych uruchamiana w kontenerze. Jeśli w pliku *appsettings-custom.json* dla serwisu *tenant-api* podano ciąg połączenia do bazy danych na serwerze zainstalowanym lokalnie, należy usunąć tę opcję z polecenia.

2.9.2. Pierwszy tenant

Po pomyślnym uruchomieniu kontenerów, należ wpisać w przeglądarce adres domeny środowiska nAxiom, np. *https://nax-deploy.local/tenantsadmin*. Zostanie wyświetlone okno logowania do aplikacji *TenantAdminSPA*.

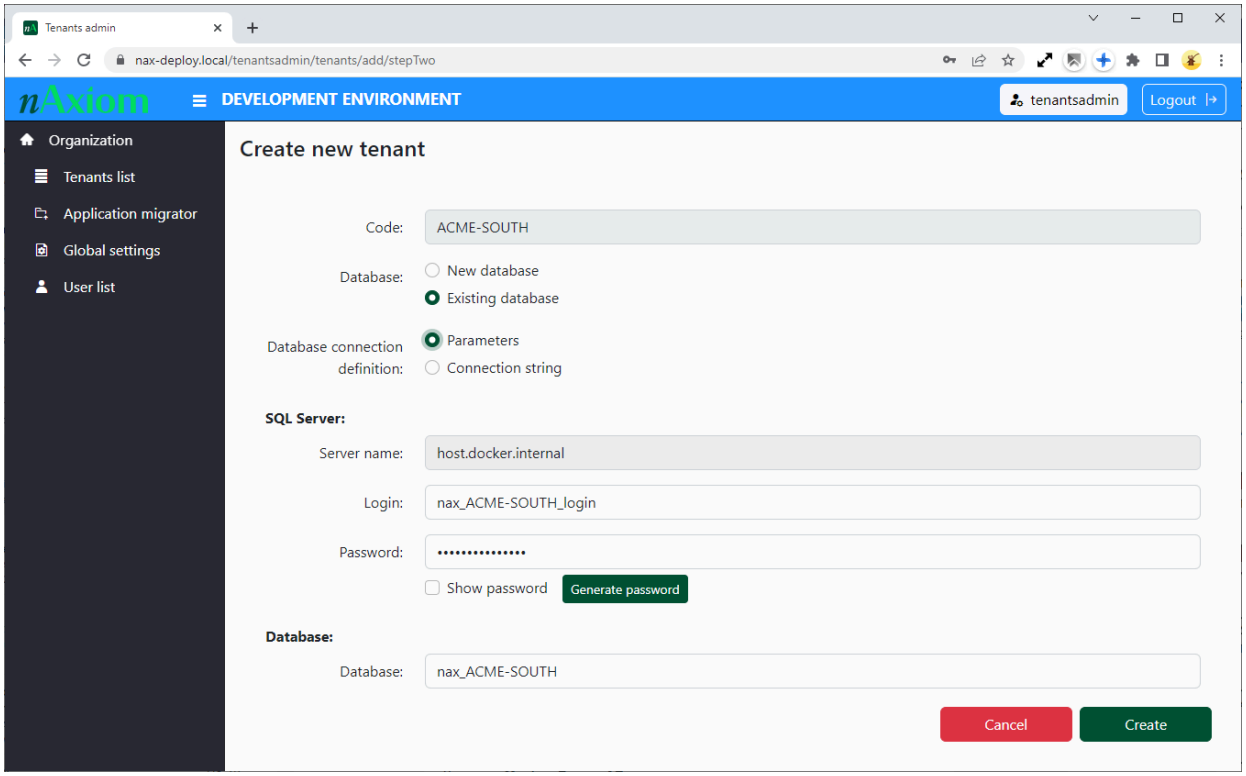


Logowanie do TenantAdminSPA

Standardowe dane logowania to login *tenantsadmin* i hasło *!Q2w3e4r%T*. Po zalogowaniu zostanie wyświetlona strona aplikacji z menu po lewej stronie i listą tenantów w części głównej. Aplikację opisano w osobnej dokumentacji. W celu utworzenia pierwszego tenanta należy kliknąć przycisk `Add new tenant` i podawać dane w uruchomionym kreatorze.

- `Code`: kod tenanta; tylko znaki alfanumeryczne, znaki podkreślenia (_) i myślniki (-); kodu tenanta określonego w tym kroku nie można zmienić.
- `Database`:
  - `New database`: dla tworzonego tenanta zostanie utworzona nowa baza danych o podanych parametrach.
  - `Existing database`: tenant zostanie powiązany z istniejącą bazą danych o podanych parametrach.
- `Database connection definition`:

- `Parameters`: (domyślnie) należy podać parametry połączenia z bazą danych w polach:
  - `Server name`: adres serwera SQL Server; w przypadku gdy jest zainstalowany lokalnie na tej samej maszynie *host.docker.internal* (*host.minikube.internal* dla klastra testowego *minikube*).
  - `Login`: Login do bazy danych; jeśli nie istnieje zostanie utworzony, zaleca się podanie loginu utworzonego z mocnym hasłem specjalnie dla administratora tenantów.
  - `Password`: Hasło do bazy danych (dostępny jest przycisk generowania mocnego hasła).
  - `Database`: Nazwa bazy danych pierwszego tenanta
- `Connection string`: należy podać ciąg połączenia w polu tekstowym `Database connection string`.



Tworzenie pierwszego tenanta

Kliknij przycisk `Create`, aby utworzyć pierwszego tenanta w środowisku nAxiom. Wymagane jest podwójne potwierdzenie. Nowy tenant jest tworzony jako nieaktywny. Po jego utworzeniu należy go aktywować. Dodatkowo konieczne jest podanie wartości w polu `Customer Name` na karcie `Main`. Jest to ciąg znaków, na podstawie którego jest identyfikowana licencja danego tenanta (podczas pierwszego logowania konsultant tego tenanta musi zaimportować plik licencji zgodny z wartością w polu `Customer Name`).

Należy zauważyć, że wartość podana w polu `Code` zostaje domyślnie użyta jako przedrostek adresu URL do nAxiom dla danego tenanta. Jeśli tenant ma używać bazowego adresu środowiska nAxiom bez przedrostka, należy wyczyścić pole `Url prefix` na karcie `Configuration`.
Po dodaniu tenanta konieczny jest restart wszystkich serwisów środowiska nAxiom.
Teraz można przejść do opisu uruchamiania kompletnego środowiska nAxiom. Należy pamiętać o przywróceniu prawidłowej konfiguracji serwera nginx (w pliku konfiguracyjnym wymagane są wpisy dla wszystkich serwisów nAxiom).

2.9.3. Udostępnienie plików konfiguracyjnych z istniejącej instalacji

nAxiom oferuje możliwość automatycznego utworzenia pierwszego tenanta na podstawie plików konfiguracyjnych *appsettings.json* dla wszystkich serwisów pochodzących z istniejącej instalacji. Ten scenariusz jest używany w przypadku aktualizacji środowiska nAxiom z wersji bez obsługi tenantów. Folder *toMigrate* jest automatycznie tworzony przez instalator. Następnie, podczas pierwszego uruchomienia serwis *tenant-api* tworzy pierwszego tenanta. W przypadku wdrażania nAxiom z obrazów Docker wymagane jest samodzielne zebranie tych plików, zmiana ich nazw i umieszczenie ich w katalogu *toMigrate*.
Katalog *toMigrate* jest montowany w kontenerze serwisu *tenant-api* zgodnie z deklaracją w pliku *docker-compose-volumes.yml*:

```
  tenant-api:
    volumes:
      ...
      ...
      - '${NAX_ROOT_CONF_DIR}/toMigrate/:/home/app/toMigrate:ro'


...
```

Folder *toMigrate* musi zawierać następujące pliki:

- *api_appsettings-protected.json*
- *api_appsettings.json*
- *auth_appsettings-protected.json*
- *auth_appsettings.json*
- *docapi_appsettings-protected.json*
- *docapi_appsettings.json*
- *mobileapi_appsettings-protected.json*
- *mobileapi_appsettings.json*

- *ocrapi_appsettings-protected.json*
- *ocrapi_appsettings.json*
- *publicapi_appsettings-protected.json*
- *publicapi_appsettings.json*
- *reportsapi_appsettings-protected.json*
- *reportsapi_appsettings.json*
- *syncfusion_appsettings.json*
- *taskservice_appsettings-protected.json*
- *taskservice_appsettings.json*
- *tenantsapi_appsettings-protected.json*
- *tenantsapi_appsettings.json*

Jeśli w katalogu głównym wdrożenia nAxiom będzie się znajdował katalog *toMigrate* z wymaganymi plikami, podczas uruchamiania kontenerów zostanie utworzony pierwszy tenant i środowisko nAxiom będzie działać prawidłowo.

2.10. Uruchomienie środowiska nAxiom

Przykładowe polecenie uruchamiające pełen zestaw kontenerów nAxiom, serwer bazy danych i serwer proxy wygląda następująco:

```
docker compose \
    -f docker-compose.yml \
    -f docker-compose-common.yml \
    -f docker-compose-extra.yml \
    -f docker-compose-volumes.yml \
    --env-file docker.env \
    --profile full \
    --profile db \
    --profile proxy \
    up -d
```

2.11. Wdrożenie nAxiom w klastrze Kubernetes

Na potrzeby wdrożenia nAxiom w klastrze Kubernetes należy zdefiniować obiekty *Deployment* i *Service* dla każdego serwisu nAxiom. Zalecane jest powołanie osobnych podów dla każdego serwisu nAxiom.
W celu nadpisania ustawień domyślnych należy skorzystać z plików *appsettings-custom.json* i *config-custom.json* i dostarczyć je do podów, używając obiektów *ConfigMap* lub *Secret*.

Poniżej zamieszczono przykładowe definicje obiektów *Deployment* i *Service* dla serwisu *auth*.

> Uwaga: W obecnej wersji serwis *TaskService* może być działać wyłącznie w pojedynczej replice.

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-deployment
  namespace: naxiom
spec:
  replicas: 1
  selector:
    matchLabels:
      app: auth
  template:
    metadata:
      labels:
        app: auth
    spec:
      containers:
        - name: auth
          image: auth:1.11.0.0
          imagePullPolicy: IfNotPresent
          env:
            - name: IMG_DEPLOY_TYPE
              value: K8S
            - name: IMG_DEPLOY_NAMESPACE
              value: naxiom
            - name: IMG_DOMAIN_NAME
              value: k8s.nax-deploy.com
          volumeMounts:
            - name: appsettings-volume
              mountPath: /home/app/appsettings-custom.json
              subPath: appsettings-custom.json
```

```
            readOnly: true
          - mountPath: /usr/local/share/ca-certificates/certificate.crt
            name: cert
            subPath: certificate.crt
            readOnly: true
          - mountPath: /usr/local/share/ca-certificates/ca.crt
            name: ca-cert
            subPath: ca.crt
            readOnly: true
          - mountPath: /root/.aspnet/https
            name: key
            readOnly: true
          - mountPath: /etc/krb5.keytab
            name: keytab
            subPath: krb5.keytab
            readOnly: true
      volumes:
        - name: appsettings-volume
          configMap:
            name: auth-config
            optional: true
        - name: cert
          secret:
            secretName: cert-naxiom-internal
            optional: false
            items:
              - key: tls.crt
                path: certificate.crt
        - name: key
          secret:
            secretName: cert-naxiom-internal
            optional: false
            items:
              - key: tls.key
                path: certificate.key
        - name: ca-cert
          secret:
```

```
            secretName: cert-naxiom-internal
            optional: false
            items:
              - key: ca.crt
                path: ca.crt
        - name: keytab
          secret:
            secretName: auth-keytab
            optional: true
            items:
              - key: krb5.keytab
                path: krb5.keytab
```

Service

```
apiVersion: v1
kind: Service
metadata:
  name: auth-service
  namespace: namespace
spec:
  selector:
    app: auth
  type: ClusterIP
  ports:
    - protocol: TCP
      port: 5001
      targetPort: 5001
```

3. Konfiguracja zaawansowana

3.1. Własne pliki ustawień

Użytkownik może zdefiniować własne ustawienia dla serwisów nAxiom, umieszczając pliki konfiguracyjne w podkatalogu *configs* w katalogu głównym wdrożenia. Pliki te muszą znajdować się w katalogach

o nazwach zgodnych z nazwami serwisów. Serwisy dotnetowe wymagają przygotowania plików *appsettings-custom.json*, natomiast serwisy angularowe plików *config-custom.json*.
Na przykład, aby zmienić rodzaj środowiska nAxiom z domyślnego *deweloperskiego* na *produkcyjne*, należy w ścieżce *NAX_ROOT_CONF_DIR/configs/api* umieścić plik *appsettings-custom.json* z następującym wpisem:

```
{
  "AppConfiguration": {
    "Environment": "Production"
  }
}
```

W przypadku wdrożenia w klastrze Kubernetes powyższe pliki mogą być dystrybuowane do podów poprzez mechanizm obiektów *ConfigMaps* lub *Secrets*.

3.2. Parametry konfiguracyjne

W plikach *appsettings.json* i *config.json* poszczególnych serwisów znajdują się parametry konfiguracyjne, które użytkownik może zmieniać. W przypadku wdrożenia kontenerów dockerowych, parametry te są ustawiane automatycznie przez skrypt na podstawie ustawień w pliku z konfiguracją środowiska *.env* (`DOMAIN_NAME` i `DEPLOY_TYPE`) oraz w plikach *docker-compose*.

3.2.1. Ustawienia w plikach appsettings.json

Sekcja `AppConfiguration`:

- `IdentityServerUrl`: adres URL serwisu *auth*.
- `ApiServerUrl`: adres URL serwisu *api*.
- `FrontAppUrl`: adres URL serwisu *front*.
- `AdminAppUrl`: adres URL serwisu *admin*.
- `TenantsApiUrl`: adres URL serwisu *tenant-api*.
- `WorkflowUrl`: adres URL serwisu *workflow*.
- `TaskServiceServerUrl`: adres URL serwisu *taskservice*.
- `TelerikReportsServerUrl`: adres URL serwisu *telerik-reports*.
- `PublicApiServerUrl`: adres URL serwisu *public-api*.
- `SyncfusionApiServerUrl`: adres URL serwisu *syncfusion-api*.
- `IdentityServerExternalUrl`: zewnętrzny adres URL do serwisu *auth*.
- `PathBase`: musi mieć wartość */docapi* dla serwisu *documentation* u */reportsapi* dla serwisu *telerik-reports*, aby kierować ruch przez serwer proxy.

Sekcja `AppConfiguration/MultiTenancyConfig`:

- `LoadTenantConfigurationFromService`: musi mieć wartość *true*, aby była włączona obsługa wielu tenantów.
- `BaseUrl`: bazowy adres URL wdrożenia nAxiom.

Sekcja `AppConfiguration/OriginConfiguration`:

- `IsHostedOnOnePort`: musi mieć wartość *true*, jeśli ruch jest przekierowywany przez serwer proxy.
- `OriginUrl`: bazowy adres URL wdrożenia nAxiom.

Sekcja `ConnectionStrings` (tylko serwis *tenant-api*):

- `SQLConnectionString`: prawidłowy ciąg połączenia do bazy danych SQL Server. Adres serwera zainstalowanego na komputerze lokalnym to *host.docker.internal* (Docker) lub *host.minikube.internal* (minikube). Wymagana jest konfiguracja obsługi połączeń zewnętrznych. Jeśli dana instancja używa portu innego niż domyślny *1433*, konieczne jest podanie tego portu w ciągu połączenia.
  W przypadku, kiedy SQL Server jest uruchamiany jako dokerowy kontener, należy używać aliasu *mssql*. Taka baza danych jest dostępna w programie SQL Server Management Studio pod adresem *localhost,1443* address.

Sekcja `AppConfiguration` (tylko serwis *tenant-api*):

- `FileStorageRoot`: główny katalog plików statycznych tenantów. Domyślnie jest to podkatalog katalogu głównego aplikacji: */home/app/nAxiom*.

3.2.2. Ustawienia w plikach config.json

- `multiTenancyConfig/baseUrl`: bazowy adres URL wdrożenia nAxiom.
- `apiServerConfig/serverUrl`: adres URL serwisu *api*.
- `identityServerConfig/serverUrl`: adres URL serwisu *auth*.
- `identityServerConfigForMobile/serverUrl`: adres URL serwisu *auth*.
- `reportsApiServerConfig/apiUrl`: adres URL serwisu *telerik-reports*.
- `reportsApiServerConfig/designerUrl`: adres URL modułu *Telerik Reports Web Designer*.
- `workflowConfig/workflowUrl`: adres URL serwisu *workflow*.
- `adminConfig/adminUrl`: adres URL serwisu *admin*.
- `documentationConfig/documentationUrl`: adres URL serwisu *documentation*.
- `publicApiServerConfig/serverUrl`: adres URL serwisu *public-api*.

- `passwordGrantIdentityServerConfig/serverUrl`: adres URL serwisu *auth*.
- `frontConfig/frontUrl`: adres URL serwisu *front*.
- `tenantApiServerConfig/serverUrl`: adres URL serwisu *tenant-api*.

3.3. Usuwanie plików ustawień

Pliki ustawień mogą zawierać dane poufne, na przykład poświadczenia logowania do bazy danych. Ze względów bezpieczeństwa zaimplementowano mechanizm, który powoduje, że takie pliki są usuwane z kontenera po ich wczytaniu podczas uruchamiania, dzięki temu żaden użytkownik kontenera nie będzie miał dostępu do tych danych.
Aby zapewnić taką ochronę, należy skonfigurować mapowanie takiego pliku dla klucza *command:* danego serwisu. W prezentowanym przykładzie odpowiednie deklaracje znajdują się w pliku *docker-compose-common.yml*, na przykład:

```
api:
  command: '/root/.aspnet/configs/appsettings-custom.json:
         /home/app/appsettings-custom.json'
```

Powyższy wpis spowoduje, że plik *appsettings-custom.json* dla serwisu *api* zostanie wczytany do katalogu */home/app* w systemie plików kontenera, a następnie usunięty — po wczytaniu ustawień z tego pliku w momencie rozruchu kontenera.
W klastrze Kubernetes można użyć tego mechanizmu w momencie definiowania obiektu *Pod*. Wymaga to przekazania odpowiednio argumentów do kontenera, na przykład:

```
args:
  - "/root/.aspnet/configs/appsettings-custom.json:
         /home/app/appsettings-custom.json")
```

3.4. Konfiguracja logowania komunikatów

Ustawienia logowania komunikatów dla kontenerów serwisów dotnetowych można zmienić dla każdego serwisu z osobna, używając pliku konfiguracyjnego *nlog-custom.config* w ścieżce *katalog-główny-wdrożenia/configs/nazwa-serwisu/*. Poniżej opisano dostępne parametry konfiguracyjne.
Dostępnych jest 6 predefiniowanych poziomów logowania:

- *TraceLevel*: wszystkie informacje z danego serwisu.
- *DebugLevel*: wszystkie informacje z danego serwisu, oprócz informacji z poziomu *TraceLevel*.
- *InfoLevel*: komunikaty informacyjne, ostrzegawcze i komunikaty o błędach.
- *WarnLevel*: komunikaty ostrzegawcze i komunikaty o błędach.

- *ErrorLevel*: tylko komunikaty o błędach.
- *OffLevel*: żadne komunikaty nie są logowane.

Komunikaty mogą być logowane w trzech miejscach docelowych (w razie potrzeby można zdefiniować własne):

- *logfile*: komunikaty są zapisywane do pliku w podkatalogu *logs* w katalogu głównym aplikacji.
- *sqlitedb*: komunikaty są zapisywane do pliku bazy danych SQLite w katalogu głównym aplikacji.
- *console*: komunikaty są wysyłane do konsoli, ten tryb jest dostępny TYLKO w przypadku kontenerów; w tym przypadku demon Docker przechwytuje te komunikaty i zapisuje je w plikach tekstowych lub dziennikach systemowych; możliwe jest skonfigurowanie oprogramowania gromadzącego i przetwarzającego dane, w rodzaju Zabbix, Elasticsearch + Fluentd + Kibana lub Splunk.

Dodatkowo są trzy zmienne określające poziom logowania dla miejsc docelowych zdefiniowanych powyżej:

- *WildcardConsoleLoggerLevel*: domyślnie *InfoLevel*, dotyczy komunikatów wysyłanych na konsolę.
- *WildcardFileLoggerLevel*: domyślnie *OffLevel*, dotyczy komunikatów zapisywanych w pliku.
- *WildcardDatabaseLoggerLevel*: domyślnie *OffLevel*, dotyczy komunikatów zapisywanych w bazie danych SQLite.

W bazowym pliku konfiguracyjnym *nlog.config* dla serwisów dotnetowych nie ma reguł filtrowania zdarzeń z loggera *Entity Framework*, ponieważ domyślnie logowane są wszystkie zdarzenia od poziomu *Info* włącznie. Takie reguły można dodać w pliku *nlog-custom.config*, jak w poniższym przykładzie.

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- set trace for console logger -->
  <variable name="WildcardConsoleLoggerLevel" value="${TraceLevel}" />

  <rules>
    <!-- and ignore microsoft and identity server errors -->
    <logger name="Microsoft.*" maxlevel="${Infolevel}" final="true" />
    <logger name="IdentityServer4.*" maxlevel="${WarnLevel}" final="true" />
  </rules>
</nlog>
```

> Ważne: dla serwisu *api* poziom logowania w bazie SQLite jest określany globalnie parametrem w aplikacji *TenantAdminSPA* (`Global settings` > `Error logging level`). Dla tego serwisu mechanizm własnego pliku konfiguracyjnego opisany powyżej nie zadziała.

3.5. Pliki custom-entrypoint.sh

Własne pliki skryptów służą do wykonywania dowolnych poleceń w momencie uruchamiania kontenerów, na przykład można w ten sposób instalować czcionki i drukarki. Własne skrypty są montowane w kontenerach przy użyciu mechanizmu *volumes*.

> Uwaga:
>
> - Pliki skryptów muszą zaczynać się od sekwencji *#!/bin/sh* (opcjonalnie *#!/bin/bash*).
> - W plikach *custom-entrypoint.sh* linie muszę kończyć się zgodnie z konwencją systemów uniksowych znakiem *LF* (*\n*). Pliki przygotowane w systemie Windows mogą mieć na końcu linii parę znaków *CRLF* (*\r\n*). W takiej sytuacji konieczna będzie konwersja pliku

3.5.1. Instalacja drukarek ZPL

Platforma nAxiom oferuje możliwość drukowania kodów kreskowych na drukarkach termotransferowych ZPL (Zebra). Aby ta funkcjonalność działała w skonteneryzowanym wdrożeniu nAxiom, podczas uruchamiania kontenerów musi być wykonywany skrypt, który instaluje taką drukarkę w systemie Linux.
W przypadku drukarki podłączonej do portu USB, należy zapewnić dostęp z kontenera do tego portu w oparciu o osobną dokumentację.
Wymagane czynności to uruchomienie systemu CUPS, dodanie użytkownika *root* do grupy *lpadmin* i dodanie drukarki Zebra dostępnej przez gniazdo pod adresem *host.docker.internal* na porcie *9100*. Zapisanie tych czynności w formie skryptu *custom-entrypoint.sh* i umieszczenie w ścieżce *katalog-główny-wdrożenia/scripts/task-service/* spowoduje, że będą one automatycznie wykonywane za każdym razem podczas uruchamiania kontenera. Poniżej przykład takiego skryptu:

```
#!/bin/sh

# run cups daemon
echo "Running CUPS daemon."
```

```
cupsd

# add lpadmin group to root user
echo "Adding lpadmin group to root user."
addgroup root lpadmin

# and add ZPL printer emulator
echo "Adding ZPL printer to the list of printers in CUPS."
lpadmin -p ZPL_Printer -E -m drv:///sample.drv/zebra.ppd -D "Zebra ZPL Printer"
        -E -v socket://host.docker.internal:9100
```

3.5.2. Ustawienia narodowe kontenera

Domyślne ustawienia narodowe kontenerów to *pl_PL*, aby je zmienić, należy żyć następującego skryptu:

```
#!/bin/sh

# set locale and regenerate it using locale-gen
echo "${SERVER_LANGUAGE}.UTF-8 UTF-8" > /etc/locale.gen
locale-gen

# set LANG environment for container
echo "LANG=${SERVER_LANGUAGE}.UTF-8" > /etc/locale.conf
echo "LANG=${SERVER_LANGUAGE}.UTF-8" > /home/app/.bashrc
```

3.6. Czcionki

Platforma nAxiom może wymagać instalacji dodatkowych czcionek do generowania raportów, plików PDF oraz do eksportowania danych do plików MS Excel. Wszystkie kontenery typu dotnet mają zainstalowane podstawowe czcionki z pakietu Microsoft Core TrueType Fonts. Pakiet zawiera następujące rodzaje czcionek:

- Andale Mono
- Arial Black
- Arial (Bold, Italic, Bold Italic)
- Comic Sans MS (Bold)
- Courier New (Bold, Italic, Bold Italic)
- Georgia (Bold, Italic, Bold Italic)

- Impact / Times New Roman (Bold, Italic, Bold Italic)
- Trebuchet (Bold, Italic, Bold Italic)
- Verdana (Bold, Italic, Bold Italic)

Jednak nie wszystkie odmiany dla każdej czcionki są dostępne. Jeśli czcionka użyta w definicji raportu będzie niedostępna, zostanie użyta inna czcionka. Może to być czcionka zapasowa (*fallback*) zdefiniowana jako zamiennik w pliku mapowania czcionek. W poniższym przykładzie zdefiniowano czcionkę Arial jako czcionkę zapasową dla czcionki Arial Black.

```
<?xml version='1.0'?>
<!DOCTYPE fontconfig SYSTEM 'fonts.dtd'>
<fontconfig>
    <match target='pattern'>
        <test qual='any' name='family' compare='eq'>
            <string>Arial Black</string>
        </test>
        <edit name='family' mode='assign' binding='same'>
            <string>Arial</string>
        </edit>
    </match>
</fontconfig>
```

Plik ustawień należy zapisać w ścieżce */etc/fonts/local.conf*. Tę czynność należy zautomatyzować za pomocą skryptu *custom-entrypoint.sh*, jak w przykładzie w kolejnej sekcji.

3.6.1. Własne czcionki

W celu zainstalowania własnych czcionek podczas uruchamiania kontenera należy przygotować skrypt w pliku *custom-entrypoint.sh* i umieścić go w odpowiednim podkatalogu katalogu głównego wdrożenia. Poniższy przykładowy skrypt instaluje czcionki z pakietu DejaVu oraz tworzy plik mapowania czcionek.

```
#!/bin/sh

# aktualizacja programu apt
apt update -y

# instalacja czcionek dejavu
apt install -y fonts-dejavu
```

```
# utworzenie pliku mapowania czcionki Arial Black na Arial
# w treści pliku skryptu nie może być znaków końca linii
# poniżej dodano je dla zwiększenia czytelności
echo "
 <?xml version='1.0'?>
 <!DOCTYPE fontconfig SYSTEM 'fonts.dtd'>
 <fontconfig>
    <match target='pattern'>
        <test qual='any' name='family' compare='eq'>
            <string>Arial Black</string>
        </test>
        <edit name='family' mode='assign' binding='same'>
            <string>Arial</string>
        </edit>
    </match>
 </fontconfig>" > /etc/fonts/local.conf

# odświeżenie pamięci podręcznej czcionek
fc-cache -f
```

Taki plik skryptu należy załadować do każdego kontenera, w którym mają być dostępne instalowane czcionki lub zdefiniowane mapowanie. Na przykład dla serwisu *telerik-reports* w pliku *docker-compose-volumes.yml* jest wpis:

```
- '${NAX_ROOT_CONF_DIR}/scripts/telerik-reports/custom-entrypoint.sh:
                                        /home/app/custom-entrypoint.sh:ro'
```

Aby zainstalować czcionki w kontenerze serwisu raportów, należy umieścić plik skryptu w ścieżce *katalog-główny-wdrożenia/scripts/*.

3.7. Pliki statyczne

W środowisku nAxiom działającym jako kontenery Docker lub w klastrze Kubernetes zaleca się przechowywanie plików statycznych w repozytoriach bazodanowych. Korzystanie z repozytoriów dyskowych jest możliwe, jednak niezalecane ze względów bezpieczeństwa.
Domyślny katalog główny do zapisu plików statycznych (załączniki, szablony, logo itp.) jest konfigurowany w pliku *appsettings-custom.json* w aplikacji *TenantAdminSPA*.

```
"AppConfiguration": {
    "FileStorageRoot": "/home/app/nAxiom"
}
```

Domyślnie jest to ścieżka */home/app/nAxiom* w systemie plików kontenera, przy czym */home/app* to katalog roboczy (wykonywalny) dla serwisów dotnetowych. Katalog ten należy zmapować na katalog w systemie plików hosta, korzystając z mechanizmu *volumes*. Dodatkowo, mapowanie to trzeba powtórzyć dla wszystkich serwisów, które muszą korzystać z tych samych zasobów plikowych. Na przykład serwis *publicapi* posiada endpoint */Attachment/DownloadAttachment*. Aby wykonać odpowiednie żądanie, serwis musi mieć dostęp do tego repozytorium plików w swoim kontenerze. Podobnie jest dla innych serwisów.
W takiej sytuacji, oprócz skomplikowanej konfiguracji, w systemie plików hosta znajdowałby się katalog z plikami statycznymi wszystkich tenantów, co mogłoby zagrażać bezpieczeństwu.
Problem ten rozwiązuje korzystanie z repozytoriów bazodanowych, które wykorzystują separację tenantów ze względu na odrębne bazy danych.

3.8. Elasticsearch

W celu skonfigurowania wyszukiwania pełnotekstowego za pomocą oprogramowania Elasticsearch w nAxiom, należy dodać następujące wpisy w sekcji *AppConfiguration* pliku *appsettings-custom.json* serwisu *api*:

```
"Modules": {
    "FullTextSearch": {
        "Enabled": true,
        "MaxQueryResults": 50,
        "ElasticSearch": {
          "Url": "https://elasticsearch-master.elastic:9200",
          "UserName": "elastic",
          "IndexPrefix": "naxiom",
          "Password": "pwd"
        }
    }
}
```

Dodatkowo w konfiguracji tenanta (*TenantAdminSPA* > `Tenants list` > `Edit` > `Configuration` > `Settings`) należy wpisać rozszerzenia nazw plików, które mają być indeksowane:

```
"Modules": {
```

```
    "FullTextSearch": {
        "IndexableFileExtensions": ["pdf", "doc"]
    }
}
```

Opis ustawień:

- *FullTextSearch.Enabled* ma mieć wartość *true*.
- *ElasticSearch.Url* to wewnętrzny adres serwisu Elasticsearch; w przypadku kontenerów Docker będzie to nazwa kontenera, we wdrożeniu w klastrze adres zgodny z przestrzenią nazw, w której zainstalowano Elasticsearch (domyślnie *elastic*).
- *ElasticSearch.UserName*: nazwa użytkownika używana do komunikacji.
- *ElasticSearch.Password*: hasło używane do komunikacji.

3.9. Redis

Aby włączyć dla serwisu dotnetowego obsługę pamięci podręcznej przez oprogramowanie Redis, należy dodać następujące wpisy w sekcji *AppConfiguration* pliku *appsettings-custom.json* tego serwisu:

```
"CacheConfiguration": {
      "EnableCache": true,
      "CacheProvider": "Redis",
      "EnableCacheInitialization": true
},
"RedisConfiguration": {
      "Server": "redis-master.redis.svc.cluster.local",
      "Port": 6379,
      "Password": "pwd"
}
```

Opis ustawień:

- *CacheConfiguration.CacheProvider* musi mieć wartość *Redis*.
- *RedisConfiguration.Server* określa wewnętrzny adres instancji *master*, w przypadku kontenerów Docker będzie to nazwa kontenera, we wdrożeniu w klastrze adres zgodny z przestrzenią nazw, w której zainstalowano Redis (domyślnie *redis*).
- *RedisConfiguration.Port*: port do komunikacji z instancją *master*. Wartość domyślna to *6379*.
- *RedisConfiguration.Password*: hasło używane podczas komunikacji z instancją *master*.

3.10. Uwierzytelnianie Windows

W przypadku kontenerów uruchamianych w systemach uniksowych uwierzytelnianie Windows może być używane tylko z protokołem Kerberos. W tym przypadku możliwe jest ponadto konfigurowanie uwierzytelniania Windows niezależnie dla każdego tenanta.
Niezbędne czynności obejmują:

- skonfigurowanie pliku *appsettings.json* dla serwisu *auth* (indywidualnie dla każdego tenanta w aplikacji *TenantAdminSPA*)
- skonfigurowanie ustawień w sekcji LDAP w aplikacji *AdminSPA*
- utworzenie i skonfigurowanie w usłudze Active Directory konta komputera, używanego do uwierzytelnienia użytkowników logujących się do nAxiom
- wygenerowanie pliku *krb5.keytab*
- opcjonalnie, wygenerowanie osobnych plików *keytab* dla każdego tenanta i scalenie ich w jeden plik.

> Uwaga: adres URL pod którym jest dostępne środowisko nAxiom musi zostać dodany do witryn zaufanych w przeglądarce z uwzględnieniem adresu każdego tenanta.

3.10.1. Konfiguracja serwisu *auth*

W aplikacji *TenantAdminSPA* należy przejść do ustawień tenanta i na karcie Configuration zmienić następujące klucze w oknie Settings:

```
{
...
    "WindowsLogin": {
      "UseWindowsLogin": true,
      ...
    },
    "UseADAuthentication": true,
    ...
}
```

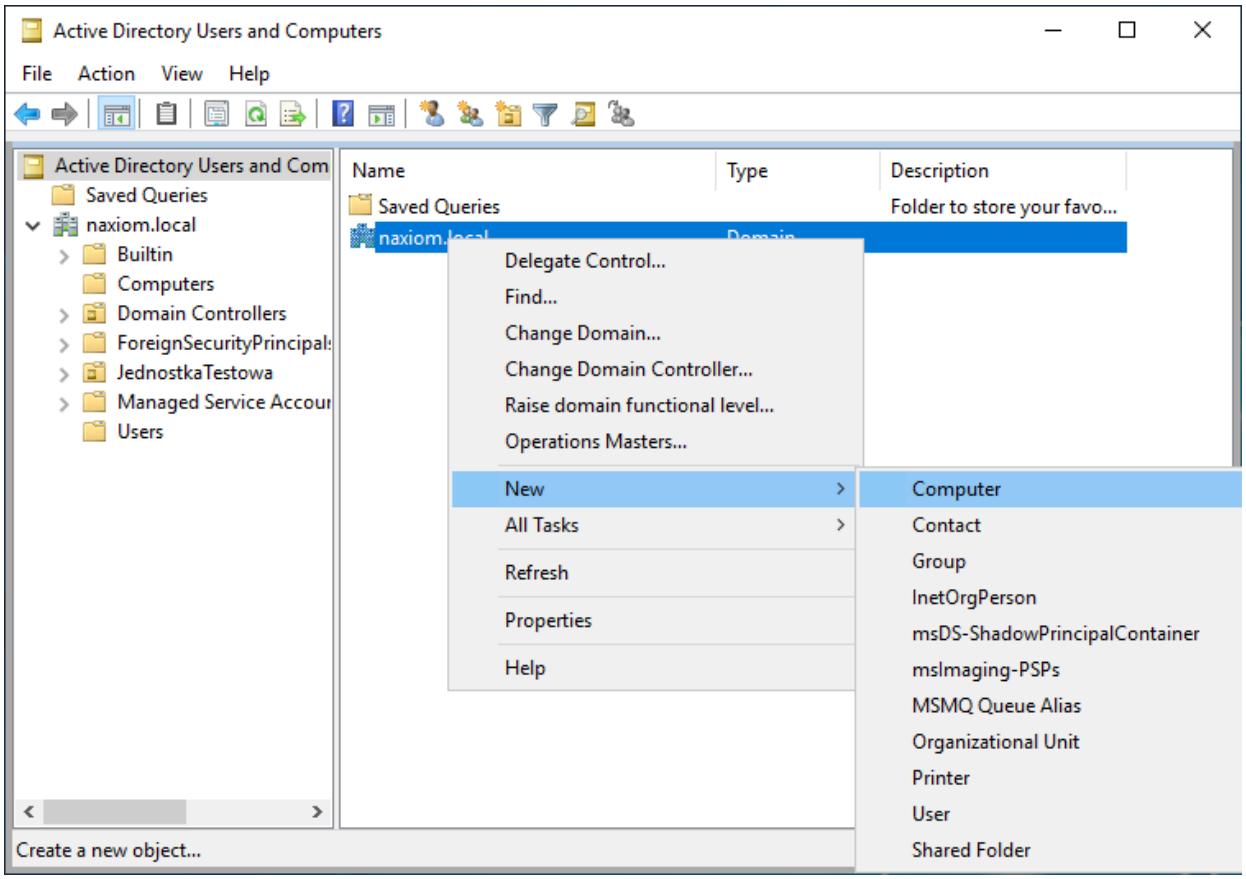
3.10.2. Konfiguracja w *AdminSPA*

W aplikacji *AdminSPA* należy skonfigurować ustawienia w sekcji LDAP (ADMINISTRACJA > Ustawienia systemu > LDAP).

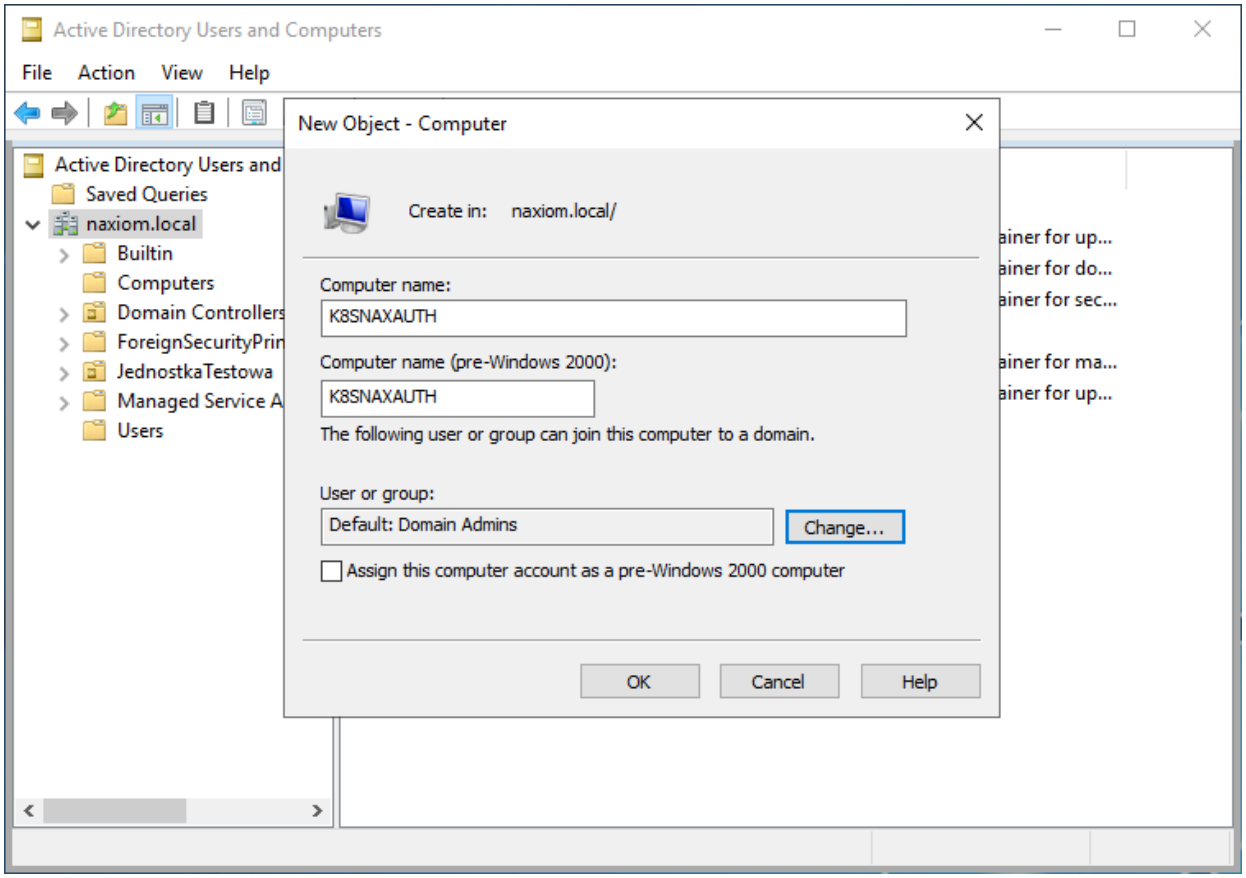
- `Ścieżka do serwera LDAP (LDAP path)`: adres kontrolera domeny Active Directory
- `Nazwa użytkownika`: nazwa użytkownika do nawiązania połączenia z kontrolerem domeny
- `Hasło użytkownika`: hasło do nawiązania połączenia z kontrolerem domeny
- `Domena LDAP`: nazwa domeny
- `Nazwa DNS lub adres IP serwera LDAP`: : adres kontrolera domeny Active Directory
- `Port do serwera LDAP`: wartość domyślna 389.
- `Połączenie przez SSL`: domyślnie wyłączone.
- `Base DN do serwera LDAP`: nazwa wyróżniająca określająca miejsca rozpoczęcia wyszukiwania na serwerze LDAP.

3.10.3. Konfiguracja w Active Directory

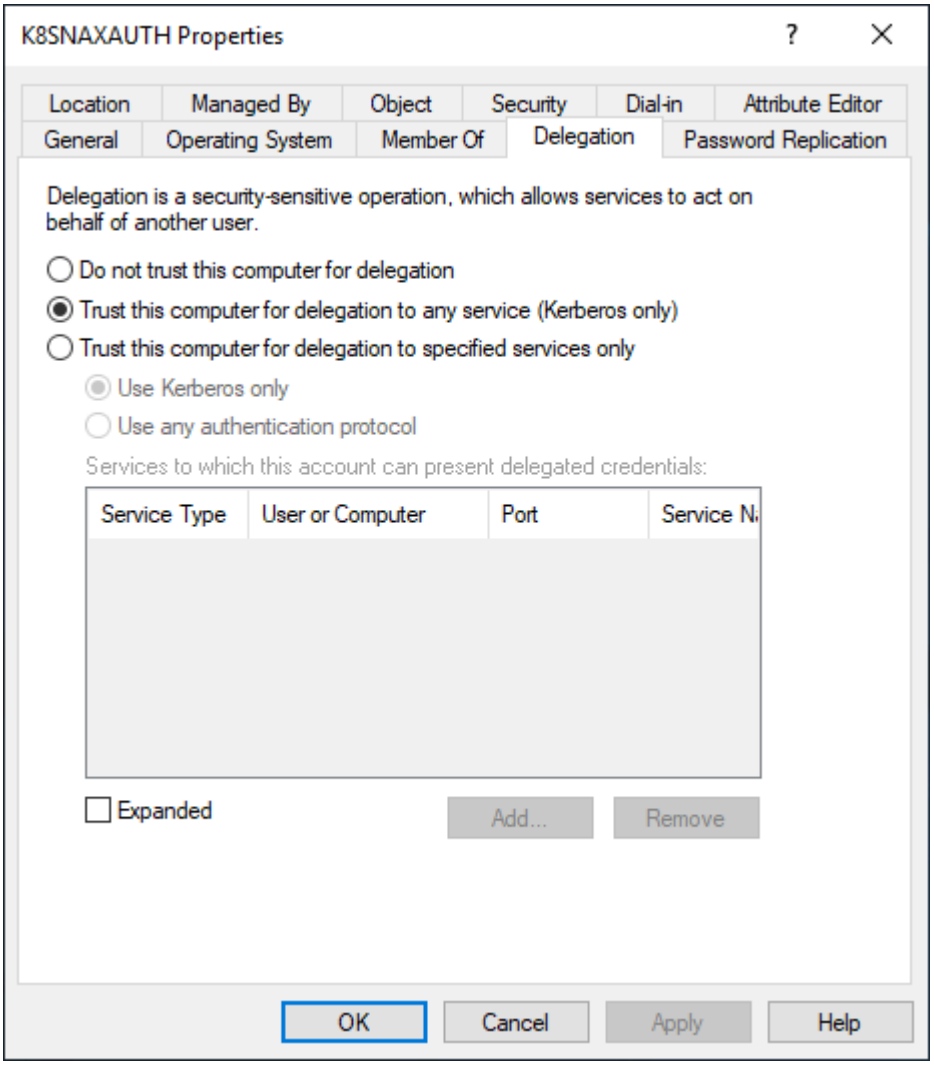
1. Zaloguj się do serwera Active Directory w swojej domenie i uruchom program Active Directory Users and Computers.
2. Kliknij prawym przyciskiem nazwę domeny i kliknij polecenie `New` > `Computer`.



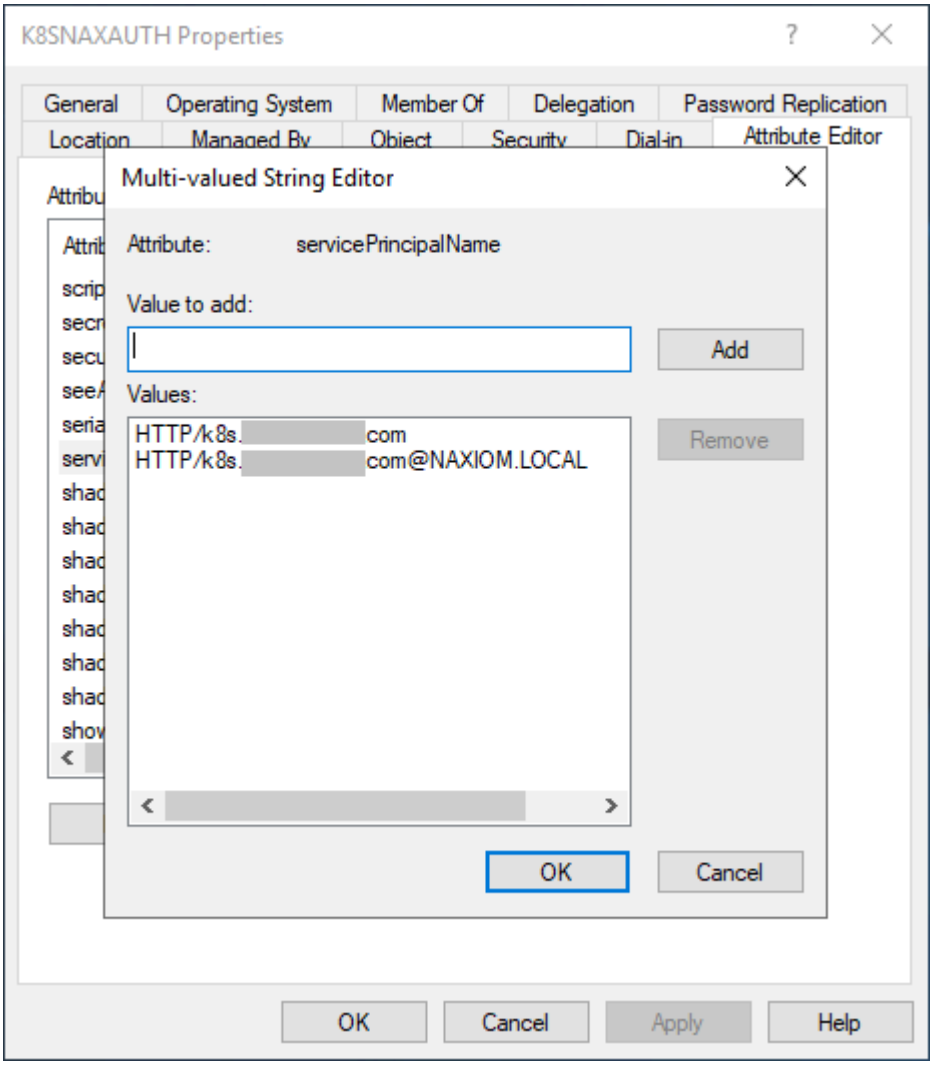
3. Wpisz nazwę w wyświetlonym oknie dialogowym i kliknij przycisk OK.



4. Wyświetl listę komputerów w domenie i wyświetl właściwości nowo utworzonego komputera.
5. Przejdź na kartę `Delegation` i zaznacz opcję `Trust this computer for delegation to any service (Kerberos only)`; sprawdź, czy w nazwie opcji jest dopisek *(Kerberos only)*.



6. Przejdź na kartę `Attribute Editor` (wymaga zaznaczenia pola wyboru `Advanced Features` w menu `View`). Znajdź na liście atrybut *servicePrincipalName* (SPN), zaznacz go i kliknij przycisk `Edit`. W wyświetlonym oknie dodaj dwie wartości: *HTTP/app.web.addr* i *HTTP/app.web.addr@DOMAIN.NAME*, gdzie *app.web.addr* to adres aplikacji, a *DOMAIN.NAME* to nazwa domeny w Active Directory.



7. Wróć na kartę `Attribute Editor`, przejdź do atrybutu *msDS-SupportedEncryptionTypes* i nadaj mu wartość 28 (1C HEX). To spowoduje włączenie szyfrowania AES256, AES128 i RC4. Aby włączyć tylko AES256, ustaw wartość 16. Więcej informacji, patrz MS techcommunity.

3.10.4. Integracja z kontenerami

W celu integracji kontenerów z protokołem Kerberos należy wygenerować plik *keytab* i zamontować go w odpowiedniej ścieżce w kontenerze serwisu *auth*. W tym celu należy uruchomić następujące polecenie na serwerze Active Directory:

```
ktpass -princ HTTP/app.web.addr@DOMAIN.NAME -pass PASSWORD
       -mapuser DOMAIN\COMPUTERNAME$ -pType KRB5_NT_PRINCIPAL
       -out C:/COMPUTERNAME.HTTP.keytab -crypto AES256-SHA1
```

- *app.web.addr*: adres przekazany do SPN
- *DOMAIN.NAME*: nazwa domeny w Active Directory
- *PASSWORD*: hasło, o ile ma być używane z plikiem *keytab*
- *COMPUTERNAME*: nazwa utworzonego komputera
- *DOMAIN*: nazwa domeny bez przyrostka.

Przykładowe polecenie z wartościami używanymi w tej sekcji:

```
ktpass -princ HTTP/webapp.devdom.local@OFFICE.LOCAL
       -pass sOm3Simpl3P4ss -mapuser OFFICE\AUTHCOMP$
       -pType KRB5_NT_PRINCIPAL
       -out C:/AUTHCOMP.HTTP.keytab -crypto AES256-SHA1
```

> Uwaga: należy pamiętać o dodaniu znaku *$* na końcu wartości atrybutu *mapuser*. Bez niego wykonanie polecenia nie powiedzie się.

Wygenerowany plik należy skopiować do ścieżki *secrets/auth* w głównym katalogu wdrożenia i zmienić jego nazwę na *krb5.keytab*. Teraz można uruchomić kontener serwisu *auth* (lub restartować, jeśli był uruchomiony).

3.10.4.1. Obsługa wielu tenantów

Opisany powyżej mechanizm logowania metodą uwierzytelniania Windows można także zastosować w przypadku środowiska z wieloma tenantami. W takiej sytuacji należy wygenerować osobny plik *keytab* dla każdego tenanta, a następnie scalić pliki tenantów w plik wynikowy *krb5.keytab*. Na przykład:

```
ktpass -princ HTTP/tenant1.devdom.local@OFFICE.LOCAL
       -pass sOm3Simpl3P4ss -mapuser OFFICE\AUTHCOMP$
       -pType KRB5_NT_PRINCIPAL
       -out C:/OFFICE.AUTHCOMP.HTTP.keytab
       -crypto AES256-SHA1
```

```
ktpass -princ HTTP/tenant2.devdom.local@CORP.LOCAL
       -pass sOm3Simpl3P4ss
       -mapuser CORP\AUTHCOMP$ -pType KRB5_NT_PRINCIPAL
       -out C:/CORP.AUTHCOMP.HTTP.keytab
       -crypto AES256-SHA1
```

Do scalenia plików należy użyć linuksowego narzędzia *ktutil* (np. z pakietu *krb5-user w Ubuntu). Działa ono interakcyjnie i wymaga użycia następującej sekwencji poleceń:

```
ktutil
read_kt OFFICE.AUTHCOMP.HTTP.keytab
read_kt CORP.AUTHCOMP.HTTP.keytab
write_kt krb5.keytab
quit
```

Plik wynikowy należy zamontować w ścieżce */etc/* kontenera serwisu *auth*, używając metody odpowiedniej dla typu wdrożenia.

4. Dodatek: Wdrożenie w Windows z użyciem skryptu startowego

W celu ułatwienia wdrożenia nAxiom z obrazów dockerowych na komputerze lokalnym z systemem Windows opracowano skrypt oraz szablony plików konfiguracyjnych, które pozwalają zautomatyzować proces wdrożenia.

Do wdrożenia nAxiom przy użyciu tej metody potrzebny jest komputer z systemem Windows z zainstalowanym środowiskiem WSL 2 oraz program Docker Desktop do zarządzania obrazami i kontenerami. Wymagane jest także wcześniejsze załadowanie obrazów nAxiom do lokalnego repozytorium, np poleceniem `docker load` (patrz Ładowanie obrazów do repozytorium).

Ta metoda obejmuje kroki, które opisano w głównej części artykułu. Różnica polega na tym, że skrypt startowy automatycznie wykonuje szereg tych kroków — modyfikacja pliku *hosts*, generowanie i instalowanie certyfikatów, uruchomienie kontenerów — na podstawie plików konfiguracyjnych przygotowanych w folderze wdrożenia.

W folderze wdrożenia musi znaleźć się co najmniej jeden podfolder o dwuczłonowej nazwie z kropką — w przykładowej konfiguracji *naxdev.local*. Ta nazwa zostanie użyta jako nazwa domeny nAxiom. Taki format nazwy jest wymagany ze względu na to, że konieczność stosowania szyfrowanych połączeń w przypadku obsługi wielu tenantów wymaga stosowania certyfikatu typu *wildcard*, a takiego certyfikatu nie można wystawić dla domeny najwyższego poziomu.

Uruchomienie skryptu z opisanymi plikami konfiguracyjnymi utworzy witrynę nAxiom korzystającą z bazy danych na serwerze SQL Server działającym w kontenerze. Jako serwer WWW i serwer działa serwer nginx również uruchomiony w kontenerze. Jeśli użytkownik utworzy podfoldery dla kilku witryn, wszystkie one będą korzystały z tego samego serwera bazy danych i serwera WWW/proxy.

4.1. Folder wdrożenia

Folder wdrożenia to folder, w którym znajdą się pliki konfiguracyjne, pliki skryptów oraz struktura podfolderów wymagana do wdrożenia. Patrz także Katalog główny wdrożenia.
Minimalna zawartość folderu wdrożenia jest następująca:

- *naxdev.local*: podfolder z nazwą domeny nAxiom; zawiera pliki konfiguracyjne w określonej strukturze podfolderów (opisane w dalszej części); utworzenie kilku takich podfolderów spowoduje wdrożenie kilku środowisk nAxiom pod różnymi adresami, analogicznie jak w przypadku instalowania kilku witryn na serwerze IIS na komputerze lokalnym.
- *.env*: plik konfiguracyjny ze zmiennymi środowiskowymi opisany w następnej sekcji.
- *start.bat*: plik wsadowy skryptu startowego; musi zostać wykonany z wiersza polecenia `cmd` systemu Windows uruchomionego z uprawnieniami administratora.

> Uwaga: Wykonanie skryptu w powłoce *PowerShell* nie powiedzie się.

- *start.sh*: skrypt powłoki *bash* wywoływany przez skrypt *start.bat* w podsystemie WSL.
- *docker-compose-starter.yml*: plik używany podczas uruchamiania kontenerów, zawiera konfigurację kontenerów *mssql* i *nginx*.

4.2. Zmienne środowiskowe

Plik *.env* zawiera wartości zmiennych środowiskowych używanych przez skrypt startowy.

```
# folder wdrożenia
NAX_ROOT_CONF_DIR="C:/Docker"
# tag obrazu SQL Server (profil wdrożenia: db)
# dla Azure SQL Edge "latest"
NAX_MSSQL_VERSION="2019-latest"
# katalog plików bazy danych (kontener mssql)
NAX_MSSQL_DATA_DIR="C:/Docker/mssql"
# hasło do serwera mssql
NAX_MSSQL_PASSWORD="p@sswOrd"
# port kontenera mssql, domyślnie 1433
# jeśli port jest używany, ustaw inną wartość
NAX_MSSQL_EXPOSED_PORT="1443"
# tag obrazu nginx (profil wdrożenia: proxy)
NAX_NGINX_VERSION="latest"
```

> Uwaga: Plik *.env* musi mieć znaki końca linii zgodne z systemami uniksowymi, to jest *LF*, a nie *CRLF*. W przeciwnym razie wykonanie skryptu *start.sh* w podsystemie WSL nie powiedzie się.

4.3. Folder domeny

Folder domeny (standardowo *naxdev.local*) musi zawierać pliki konfiguracyjne umieszczone w odpowiedniej strukturze folderów.
Folder domeny musi zawierać następującą strukturę podfolderów oraz pliki konfiguracyjne (oznaczone kursywą):

- certs
- configs
    - admin
    - api
    - auth
    - documentation
    - front
    - mobile-api
    - public-api
    - syncfusion-api
    - task-service
    - telerik-reports
    - tenant-admin
    - tenant-api
    - workflow
- nginx
    - *naxiom.conf*
    - *naxiom-buffers.conf*
    - *naxiom-certificate.conf*
    - *naxiom-headers.conf*
- scripts
    - syncfusion-api
    - task-service
    - telerik-reports
- secrets
    - api
        - *appsettings-custom.json*

        - auth
        - tenant-api
            - *appsettings-custom.json*
    - *.env*
    - *docker-compose.yml*
    - *docker-compose-common-proxy.yml*
    - *docker-compose-extra-proxy.yml*
    - *docker-compose-volumes.yml*

Folder *nginx* powinien zawierać cztery pliki konfiguracyjne opisane w rozdziale Pliki konfiguracyjne serwera nginx.
W przypadku zamiaru użycia skryptów konfiguracyjnych wykonywanych podczas uruchamiania kontenerów, należy w folderze *scripts* utworzyć podfoldery z nazwami odpowiednich serwisów i w nich umieścić pliki skryptów. Patrz Pliki custom-entrypoint.sh.
Wymagane są dwa pliki *appsettings-custom.json*: w folderze *secrets/api* oraz *secrets/tenant-api*. Pierwszy może zawierać tylko parę nawiasów klamrowych, a drugi musi zawierać parametry połączenia z bazą danych:

```
{
  "ConnectionStrings": {
    "SQLConnectionString":
      "Server=host.docker.internal,1443;
      Initial Catalog=tenants;
      User ID=sa;
      Password=p@sswOrd;
      Connection Timeout=30;"
  },
  "AppConfiguration": {
    "FileStorageRoot": "/home/app/nAxiom"
  }
}
```

Więcej informacji o własnych ustawieniach konfiguracyjnych zawierają rozdziały:

- Własne pliki ustawień
- Parametry konfiguracyjne

Folder domeny musi zawierać plik *.env* ze zmiennymi środowiskowymi dla danej domeny nAxiom (wymagane znaki końca linii LF; patrz Uwaga):

```
# przedrostki URL dla tenantów w tej domenie
# zostaną użyte w pliku hosts do mapowania
# np. 127.0.0.1    tenant1.naxdev.local
NAX_RESERVED_TENANTS="tenant1,tenant2"
# tag obrazów nAxiom użytych we wdrożeniu
NAX_VERSION="1.11.0.0"
```

Ponadto, w folderze domeny muszą znajdować się pliki *docker-compose.yaml* z konfiguracją uruchamiania kontenerów. Pliki te zostały opisane w rozdziale Pliki docker compose.

4.4. Uruchomienie skryptu

Uruchom wiersz polecenia systemu Windows (`cmd`) z uprawnieniami administratora. Przejdź do folderu wdrożenia i wpisz:

```
C:\Docker>start.bat
```

Potwierdź pytanie o zamiar utworzenia domeny i poczekaj na ukończenie działania skryptu.
Podczas działania skrypt utworzy podfolder *.keep* w folderze wdrożenia. W razie problemów z witryną lub w celu ponownego wdrożenia witryny należy usunąć ten folder wraz z zawartością.
Po zakończeniu działania skryptu konieczne jest ponowne uruchomienie przeglądarki w celu wczytania nowych certyfikatów.

> Uwaga: Certyfikaty nie będą dostępne w przeglądarce Firefox, która nie korzysta z magazynu certyfikatów Windows.

Po pierwszym uruchomieniu skryptu dostępny będzie tylko serwis *tenant-admin* (*https://naxdev.local/tenantsadmin*). Należy go uruchomić i utworzyć pierwszego tenanta (używając przedrostka zdefiniowanego w pliku *.env* w folderze domeny). Następnie trzeba ponownie uruchomić skrypt lub zrestartować wdrożenie w programie Docker Desktop, aby zostały uruchomione wszystkie serwisy środowiska nAxiom.