Instrukcja instalacji aplikacji CMS ActiveContent dla XYZ

Spis treści

Cel dokumentu

Niniejszy dokument stanowi opis procedur awaryjnych aplikacji ACN.

Dokument zakłada, że czytelnik posiada podstawowe umiejętności z zakresu administrowania systemami Linux klasy Enterprise, w tym w szczególności systemem Red Hat Enterprise Linux.

Historia zmian

Wersja	**Data**	**Autor**	**Zmiana**
1.0	2018-04-23	Barbara Tokar	Pierwsza wersja instrukcji
1.1	2018-06-15	….	...

Słownik zmiennych

W poniższej instrukcji używane są zmienne, które należy zamienić na odpowiednie wartości podczas wykonywania poleceń.

[env]	środowisko (np. dev, uat, preprod, prod)
[home]	lokalizacja katalogu domowego użytkownika dedykowanego aplikacji ACN
[version]	wersja aplikacji
[db]	serwer, na którym uruchomiona jest baza danych
[app]	serwer, na którym uruchamiana jest aplikacja

Przygotowanie serwerów

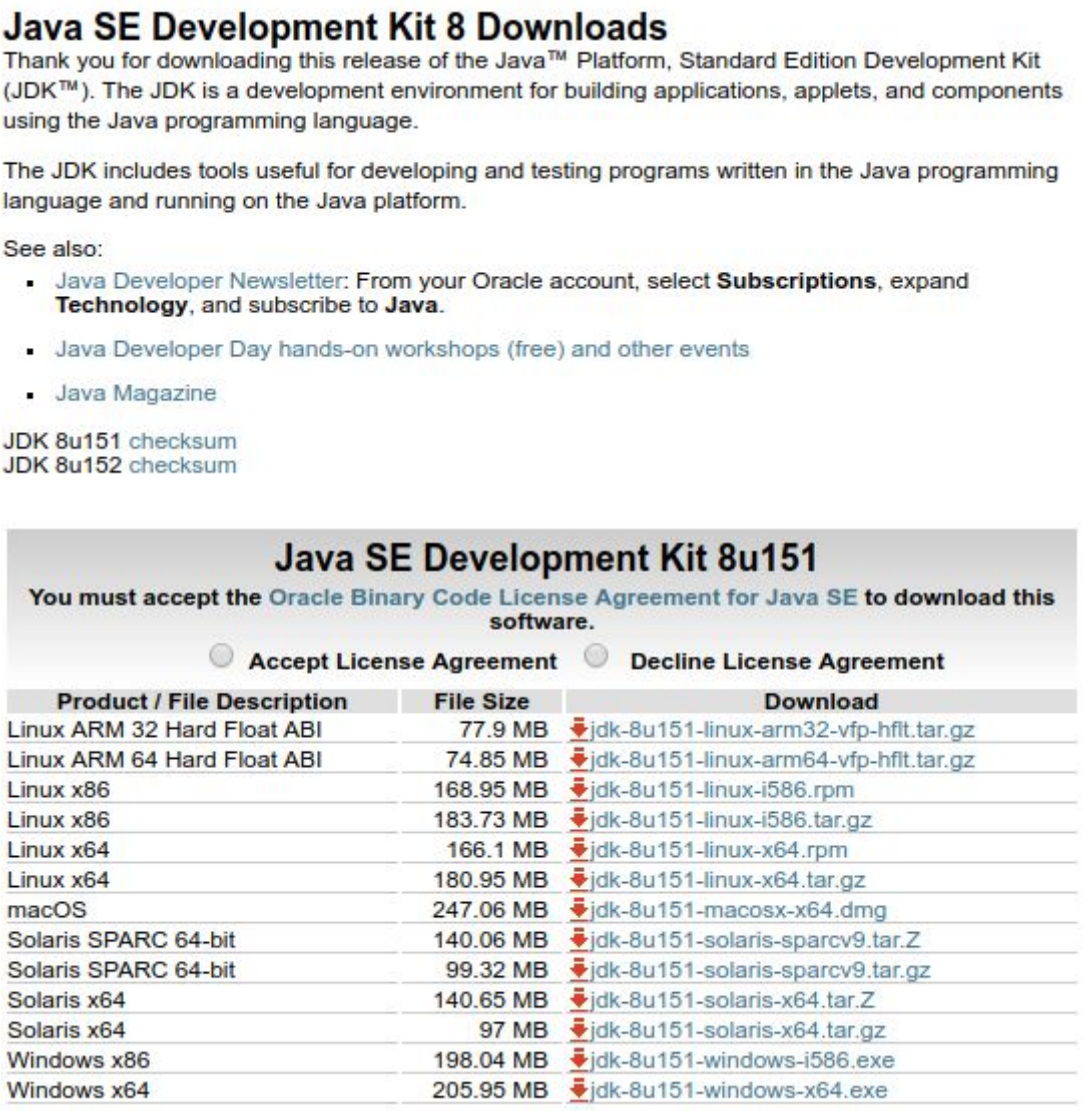
Serwery aplikacyjne

Wymagane oprogramowanie

Java 8

Do prawidłowego działania aplikacji niezbędna jest java w wersji 8. Można ją pobrać ze strony:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html



Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 8u151 checksum
JDK 8u152 checksum

Java SE Development Kit 8u151

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.9 MB	jdk-8u151-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.85 MB	jdk-8u151-linux-arm64-vfp-hflt.tar.gz
Linux x86	168.95 MB	jdk-8u151-linux-i586.rpm
Linux x86	183.73 MB	jdk-8u151-linux-i586.tar.gz
Linux x64	166.1 MB	jdk-8u151-linux-x64.rpm
Linux x64	180.95 MB	jdk-8u151-linux-x64.tar.gz
macOS	247.06 MB	jdk-8u151-macosx-x64.dmg
Solaris SPARC 64-bit	140.06 MB	jdk-8u151-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.32 MB	jdk-8u151-solaris-sparcv9.tar.gz
Solaris x64	140.65 MB	jdk-8u151-solaris-x64.tar.Z
Solaris x64	97 MB	jdk-8u151-solaris-x64.tar.gz
Windows x86	198.04 MB	jdk-8u151-windows-i586.exe
Windows x64	205.95 MB	jdk-8u151-windows-x64.exe

Rys. 1 Widok strony www.oracle.com.

Pobrane archiwum z javą należy rozpakować i umieścić w dogodnej lokalizacji na serwerze. Na serwerach środowisk DEV/UAT znajduje się w: `/usr/lib/jvm/`.

```
[user@osx02100:/usr/lib/jvm]$ ls -l
razem 0
drwxr-xr-x 3 root root 17 2017-07-14
java-1.8.0-openjdk-1.8.0.141-2.b16.el7_4.x86_64
lrwxrwxrwx 1 root root 21 2017-08-25  jre -> /etc/alternatives/jre
lrwxrwxrwx 1 root root 27 2017-08-25  jre-1.8.0 -> /etc/alternatives/jre_1.8.0
```



```
lrwxrwxrwx 1 root root 35 2017-08-25  jre-1.8.0-openjdk ->
/etc/alternatives/jre_1.8.0_openjdk
lrwxrwxrwx 1 root root 51 2017-08-25
jre-1.8.0-openjdk-1.8.0.141-2.b16.el7_4.x86_64 ->
java-1.8.0-openjdk-1.8.0.141-2.b16.el7_4.x86_64/jre
lrwxrwxrwx 1 root root 29 2017-08-25  jre-openjdk ->
/etc/alternatives/jre_openjdk
```

Rys. 2 Lokalizacja javy 8 na serwerze osx02100.

JBoss EAP 7.1

Do uruchomienia aplikacji ACN niezbędny jest serwer aplikacyjny JBoss EAP 7.1

Folder `jboss-eap-7.1` należy umieścić w katalogu `/u01/`. Do katalogu `/u01/<jboss_version>/modules` kopiujemy zawartość katalogu `/<cluster_manager_root>/optional_modules.`
Na serwerach DEV/UAT pliki binarne serwera zostały umieszczone w katalogu `/opt/`.

```
[user@osx02100:/opt]$ ls -l
razem 225560
drwx------.  2 informix   informix     62 2017-01-05  informix
drwxr-xr-x   8 mchadaj    UXUSER       188 2016-04-01  jboss-5.1.0.GA
drwxrwxr-x  10 mhryszczyk UXUSER       221 2016-04-18  jboss-eap-7.0
drwxrwxr-x  11 mateuskaminski UXUSER          238 11-03 16:05 jboss-eap-7.1
drwxr-xr-x   8 root       root         192 2015-04-21  libreoffice4.3
drwxr-xr-x   4 root       root         48 2015-04-21
LibreOffice_4.3.7.2_Linux_x86-64_rpm
```

Rys. 4 Widok zawartości katalogu /opt na serwerze osx02100.

Rsync

`Rsync` służy do synchronizacji katalogu serwera aplikacyjnego zawierającego statyczne strony http z katalogiem na serwerze http. Instalujemy poprzez:

```
sudo yum install rsync
```

Pierwszym ważnym zastosowaniem `rsync` w aplikacji ACN jest kopiowanie niezbędnych plików w trakcie deploy'u aplikacji, podczas wywołania dowolnej z operacji narzędzia `clusterManager` (`deploy, cfe`).
Przykład użycia:

```
./clusterManager.py -o cfe -c profile-acn-projectcode-dev.py
```

Wynik działania:


```
===> Creating full environment from scratch
===> PROGRESS: Preparing files with configuration.
===>  installation instruction for projectcode-dev-Cluster
Generated on Wed Apr 18 12:11:46 2018 by jkowalski
ssh HOST
su - projectcode_dev

*******************************************************
******* On APPLICATION servers: ['osx02100', 'osx02101'] *******
cd /u01/app/projectcode_dev
cp ~jkowalski/projectcode-dev/projectcode-dev-Cluster-dist.tar
~jkowalski/projectcode-dev/projectcode-dev-Cluster-install.py .
./projectcode-dev-Cluster-install.py install

*******************************************************
******* On HTTP servers: [] *******
* Create production/bin
mkdir -p ./production/bin

===> PROGRESS: Preparing archive with configuration.
===> PROGRESS: Uploading files to server.


 >>> sending incremental file list
 >>> projectcode-dev-Cluster-osx02100.tar
 >>>   24,053,760 100%    1.28MB/s     0:00:17 (xfr#1, to-chk=0/1)
 >>>
 >>> sent 21,305,411 bytes  received 34 bytes  1,039,290.00 bytes/sec
 >>> total size is 24,053,760  speedup is 1.13


 >>> sending incremental file list
 >>> projectcode-dev-Cluster-install.py
 >>>         21,411 100%  0.00kB/s     0:00:00 (xfr#1, to-chk=0/1)
 >>>
 >>> sent 5,578 bytes  received 34 bytes  3,741.33 bytes/sec
 >>> total size is 21,411  speedup is 3.82
===> PROGRESS: Preparing archive with configuration.
===> PROGRESS: Uploading files to server.


 >>> sending incremental file list
 >>> projectcode-dev-Cluster-osx02101.tar
 >>>         1,730,560 100%   36.80MB/s0:00:00 (xfr#1, to-chk=0/1)
 >>>
 >>> sent 1,248,312 bytes  received 34 bytes  832,230.67 bytes/sec
 >>> total size is 1,730,560  speedup is 1.39
```



```
>>> sending incremental file list
>>> projectcode-dev-Cluster-install.py
>>>         21,411 100%  0.00kB/s     0:00:00 (xfr#1, to-chk=0/1)
>>>
>>> sent 5,578 bytes  received 34 bytes  3,741.33 bytes/sec
>>> total size is 21,411  speedup is 3.82
CFE - success
```

Drugim zastosowaniem aplikacji `rsync` jest synchronizacja logów apache między serwerem http a serwerem aplikacyjnym. W ramach rozwoju statystyk plików będzie uruchamiany skrypt, który czyta logi apache i na ich podstawie dostarcza informacji o liczbie pobrań plików. Skrypt będzie uruchamiany z poziomu aplikacji, dlatego plik z logiem znajdujący się na serwerze http musi być zsynchronizowany ze swoim odpowiednikiem na serwerze aplikacyjnym. Skrypt synchronizujący zawartość pliku będzie uruchamiany co określony przedział czasu, wykorzystując cykliczne zadanie z crontab'a.

Wpis w crontabie na serwerze http dotyczący skryptu synchronizującego plik z logami dla środowiska DEV:

```
* * * * * /root/copyAccessLog.sh
```

Skrypt należy umieścić na serwerze http w katalogu `/root/`. Jego zawartość:

```
today=$(date +"%Y%m%d")
filename="projectcode-[env]-access.log.$today"
sshpass -p [hasło_użytkownika_projectcode_[env]] rsync
"/u01/apache_logs/projectcode_dev/$filename"
projectcode_[env]@[app]:~/access_log
```

Skrypt synchronizuje plik logów apache, z aktualnego dnia, znajdującego się na serwerze http z plikiem znajdującym się w katalogu domowym dedykowanego użytkownika OS dla danego środowiska (w tym przypadku `/home/projectcode_[env]`).

Skrypt do skanowania logów: `fileStatisticsScan.sh` znajduje się w repozytorium aplikacji ACN (`wwwxyz-app`) w katalogu `tools/`.
Plik ten należy umieścić w katalogu domowym użytkownika OS dla danego środowiska. Dla środowiska DEV jest to `/home/projectcode_dev`. Skrypt będzie uruchamiany przez aplikację co określony przedział czasu, określony w crontabie danego użytkownika.

Użytkownicy

W celu uruchomienia aplikacji ACN na serwerze aplikacyjnym musi być przygotowany użytkownik dedykowany o nazwie `projectcode_[nazwa_środowiska]`, gdzie `[środowisko]` oznacza

skrót nazwy przygotowywanego środowiska (`dev, uat, preprod, prod`). Instalacja aplikacji będzie wykonywana po zalogowaniu na serwer na dedykowanego użytkownika. Dla środowiska DEV i UAT zostały przygotowane następujące konta o nazwach:

- `projectcode_dev,`
- `projectcode_uat`.

W celu sprawdzenia poprawności utworzenia użytkownika należy zalogować się na serwer aplikacyjny poprzez:

```
ssh [nazwa_serwera]
```

, a następnie na danego użytkownika wykonując polecenie

```
su - [nazwa_uzytkownika]
```

```
ssh osx02100
Red Hat Enterprise Linux Server release 7.3

************************************************************
*                                                          *
*                 logowanie LDAP/AD/SSSD                   *
*                                                          *
************************************************************

Last login: Wed Apr 18 11:15:18 2018 from 10.6.76.82
[user@osx02100:~]$ su - projectcode_dev
Hasło:
Ostatnie logowanie: wto kwi 17 20:04:01 CEST 2018 na pts/4
[projectcode_dev@osx02100:~]$
```

Rys. 5 Logowanie na serwer aplikacyjny osx02100 na użytkownika projectcode_dev.

Użytkownik powinien posiadać własny crontab. Część funkcjonalności aplikacji realizowana jest w formie cyklicznych zadań uruchamianych przy wykorzystaniu crontab'a.

Serwer HTTP

Wymagane oprogramowanie

Apache

Do prawidłowej komunikacji serwera http z aplikacją wymagany jest Apache z zainstalowanym modułem `mod_jk`. Konfiguracja Apache obejmuje edycję pliku `vhosts.conf` z katalogu `/etc/httpd/conf.d/` oraz `workers.properties` z katalogu `/etc/httpd/conf/`. Poniższe zmiany dotyczą konfiguracji środowiska DEV dla ACN.

```
<VirtualHost *:80>
        DocumentRoot /var/www/html/projectcode
        ServerName services-projectcode-dev.xyz.pl
        ServerAlias
        UseCanonicalName Off

        RewriteEngine on
        RewriteRule ^(.*) https://%{SERVER_NAME}$1 [R,L,NC]
</VirtualHost>

<VirtualHost *:443>
        DocumentRoot /var/www/html/projectcode
        ServerName services-projectcode-dev.xyz.pl
        ServerAlias
        UseCanonicalName Off

        ErrorLog "|/usr/sbin/rotatelogs -l
/u01/apache_logs/projectcode_afrefservices_dev/services-projectcode-dev-error.log.%Y%m%d 86400"
        CustomLog "|/usr/sbin/rotatelogs -l
/u01/apache_logs/projectcode_afrefservices_dev/services-projectcode-dev-access.log.%Y%m%d 86400"
full

        ErrorDocument 403 /_error/403.html
        ErrorDocument 404 /_error/404.html
        ErrorDocument 405 /_error/405.html
        ErrorDocument 500 /_error/500.html
        ErrorDocument 503 /_error/503.html
        ErrorDocument 504 /_error/504.html

        SSLEngine on
        SSLProtocol all -SSLv2 -SSLv3
        SSLCipherSuite ALL:!SSLv2:!LOW:!ADH:!AECDH:!EXPORT:+HIGH:+MEDIUM:!RC4
        SSLOptions +StdEnvVars
        SSLHonorCipherOrder on
        Header always set Access-Control-Allow-Methods "POST, GET, OPTIONS"
        BrowserMatch "MSIE [2-6]" nokeepalive ssl-unclean-shutdown downgrade-1.0 force-response-1.0
        BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown

        SSLCertificateFile    /etc/httpd/ssl/ANY.projectcode.xyz.pl.crt
        SSLCertificateKeyFile /etc/httpd/ssl/ANY.projectcode.xyz.pl.key
        SSLCACertificateFile  /etc/httpd/ssl/endorCA.crt
```



```
        JkMount /  afrefservice-projectcode-dev
        JkMount /* afrefservice-projectcode-dev
</VirtualHost>
```

Rys. 6 Zawartość pliku konfiguracyjnego vhosts.conf.

W pliku należy zdefiniować dwa elementy VirtualHost, jeden dla standardowego portu http 80, drugi dla https 443. W `ServerName` należy ustawić główną domenę, która będzie zawarta w standardowych poddomenach aplikacji. ACN obsługuje cztery poddomeny, które należy dodać w sekcji `ServerAlias` obydwu VirtualHost'ów:

- `admin-[ServerName]`,
- `*-user-[ServerName]`,
- `crontasks-[ServerName]`,
- `time-[ServerName]`.

W `DocumentRoot` powinna znaleźć się lokalizacja aplikacji na serwerze. Poprzez `ErrorDocument` należy skonfigurować ścieżki do statycznych stron błędów znajdujących się na tym serwerze. W `ErrorLog` i `CustomLog` należy ustawić odpowiednio lokalizację logów błędów aplikacji, w drugim natomiast lokalizację logów dostępu.

Dla VirtualHost'a z portem 443 dodatkowo skonfigurowane są parametry bezpiecznego połączenia. Dokładnie takie same powinny znaleźć się w konfiguracji pozostałych środowisk. Ostatnim elementem tej konfiguracji jest nazwa workerów, które należy wstawić w linii `JkMount`. Nazwa ta powinna być spójna z nazwą workerów zdefiniowaną w drugim pliku konfiguracyjnym `workers.properties`.

Poprawna konfiguracja pliku `workers.properties` dla środowiska projectcode-dev:

```
worker.list=projectcode-dev

# ---------------------- worker projectcode-dev1
worker.projectcode-dev1.port=xx001
worker.projectcode-dev1.host=10.2.xx.yy
worker.projectcode-dev1.type=ajp13
worker.projectcode-dev1.lbfactor=1
# ---------------------- worker projectcode-dev2
worker.projectcode-dev2.port=xx101
worker.projectcode-dev2.host=10.2.xx.yy
worker.projectcode-dev2.type=ajp13
worker.projectcode-dev2.lbfactor=1
# ---------------------- worker projectcode-dev3
worker.projectcode-dev3.port=xx001
worker.projectcode-dev3.host=10.2.xx.yy
worker.projectcode-dev3.type=ajp13
worker.projectcode-dev3.lbfactor=1
# ---------------------- worker projectcode-dev4
worker.projectcode-dev4.port=x101
worker.projectcode-dev4.host=10.2.xx.yy
```



```
worker.projectcode-dev4.type=ajp13
worker.projectcode-dev4.lbfactor=1
# ---------------------- Load Balancer worker
worker.projectcode-dev.type=lb
worker.projectcode-dev.balance_workers=projectcode-dev1,projectcode-dev2,projectcode-dev3,projectcod
e-dev4
worker.projectcode-dev.sticky_session=true
worker.projectcode-dev.method=B
worker.projectcode-dev.session_cookie=CMSSESSIONID
```

Rys. 7 Zawartość pliku konfiguracyjnego workers.properties.

Prefiks nazwy workerów definiujemy w profilu aplikacji (`workerPrefix=projectcode-dev`), tutaj widoczny na liście wszystkich zdefiniowanych workerów jako: `worker.list=projectcode-dev`. `worker.list` docelowo będzie zawierać prefixy workerk'ów dla wszystkich aplikacji obsługiwanych w ramach tego serwera `Apache`. Dla każdego węzła klastra aplikacji definiujemy worker o kolejnym numerze porządkowym, zaczynając od 1. Dla tego środowiska będą uruchomione cztery węzły, po dwa na serwer aplikacyjny. Dla każdego z nich definiujemy:

- **port**: tutaj przypadają dwa węzły na serwer aplikacyjny, każdy węzeł ma swoją własną wartość `portBase` standardowo węzły pomiędzy serwerami mają taki sam `portBase` (`portBase` pierwszego węzła na jednym serwerze aplikacyjnym ma taką samą wartość jak `portBase` pierwszego węzła z drugiego serwera aplikacyjnego) zaś węzły na tym samym serwerze aplikacyjnym mają wartości `portBase` odległe o 100 (z uwagi na specyfikę działania clusterManagera, który wymaga, by każdy węzeł miał zarezerwowane 100 kolejnych portów licząc od `portBase`) Wartość `portBase` należy zapamiętać, gdyż trzeba będzie ją zdefiniować w profilu aplikacji (kolejny rozdział),
- **host**: adres IP serwera aplikacyjnego,
- **type**: ajp13,
- **lbfactor**: dla każdego workera wartość 1.

Na samym końcu konfigurujemy LoadBalancer konfigurując następujące parametry:

- **type**: lb
- **balance_workers**: wpisujemy nazwy wszystkich zdefiniowanych workerów dla tej aplikacji,
- **sticky_session**: true,
- **method**: B
- **session_cookie**: CMSSESSIONID, istotne dla prawidłowego działania aplikacji.

Dodatkowo, Apache powinien być skonfigurowany w taki sposób, aby odpowiedź z aplikacji odpytanej o zasób `/_places_objects?id=branches` zawierał nagłówek

```
Access-Control-Allow-Origin: https://<adres frontu AF>
```


np. odpowiedź z `https://xyz.pl/_places_objects?id=branches` powinna zawierać nagłówek

```
Access-Control-Allow-Origin: https://af-projectcode.xyz.pl
```

Należy pamiętać o uwzględnieniu wszystkich domen kierujących na front AF.

Po skonfigurowaniu plików `vhosts.conf` i `workers.properties` należy zrestartować serwer apache:

```
apachectl restart
```

Przygotowanie struktury katalogów

Na serwerze http należy przygotować strukturę katalogów dla logów apache'a dla instalowanej aplikacji. W katalogu `/u01/` tworzymy katalog `apache_logs/[nazwa_środowiska]` poprzez wywołanie komendy:

```
mkdir -p apache_logs/[nazwa_środowiska_aplikacji]
```

Dla środowiska DEV AC komenda wygląda następująco:

```
mkdir -p apache_logs/projectcode_dev
```

Rsync

Aplikacja Rsync służy do synchronizacji katalogu serwera aplikacyjnego. Instalujemy poleceniem:

```
sudo yum install rsync
```

Zastosowanie `rsync` opisane w rozdziale Serwery aplikacyjne.

Serwer bazy danych

Wymagane oprogramowanie

EnterpriseDB 9.6

Na serwerze bazodanowym należy zainstalować bazę danych EnterpriseDB w wersji 9.6. W bazie danych należy utworzyć użytkownika dedykowanego dla danego środowiska ACN oraz utworzyć dla niego bazę danych, do której będzie miał uprawnienia dodawania własnych schematów. Użytkownik powinien mieć również wszystkie niezbędne uprawnienia do uruchomienia polecenia `pg_dump`. Poprawność utworzenia użytkownika można sprawdzić poprzez wyświetlenie zawartości tabeli `pg_user` ze schematu `pg_catalog`.

```
select * from pg_catalog.pg_user where usename = 'projectcode_ac_dev';
-[ RECORD 1 ]-----+-------------
usename           | projectcode_ac_dev
usesysid          | 41172
usecreatedb       | f
usesuper          | f
userepl           | f
usebypassrls      | f
passwd            | ********
valuntil          |
useaccountstatus  | 0
uselockdate       |
usepasswordexpire |
useconfig         |
```

Rys. 8 Użytkownik bazy danych `projectcode_ac_dev` wraz z uprawnieniami.

Do pliku konfiguracyjnego bazy danych `pg_hba.conf` niezbędne jest dodanie linii, umożliwiających komunikację pomiędzy serwerami aplikacyjnymi a bazą danych. Wpis ma postać:

```
host    [nazwa_bazy_danych]    [nazwa_użytkownika_db]    [ip-address/podsieć]
md5
```

Wpisy dla środowiska DEV ACN wyglądają następująco:

```
host    ac    projectcode_ac_dev    10.2.xx.yy/32    md5
host    ac    projectcode_ac_dev    10.2.xx.yy/32    md5
```

Ważne: w adresie IP serwerów aplikacyjnych należy uwzględnić podsieć.

Strukturę bazy danych należy odtworzyć z przygotowanego wcześniej dump'a. W tym celu należy wykonać polecenie:

```
zcat dump.gz | psql -h localhost -p xxxx -U projectcode_ac_dev ac
```

Polecenie uruchamiamy na nowo utworzonej bazie danych (bez schematów, dump utworzy niezbędne schematy i tabele).

Ważne, wersja w bazie danych powinna zgadzać się z wersją aplikacji.

Po zalaniu bazy danych dumpem należy zweryfikować zgodność wersji aplikacji z wersją w bazie danych:

- Wersja aplikacji jest zawarta w nazwie paczek EAR z aplikacją, np. **ear-admin-[VERSION].ear**
- Wersję w bazie danych sprawdzana jest poprzez wykonanie polecenia:

```
select version from master.system_version;
```

Przykładowe wykonanie skryptu:

```
acn=> select version from master.system_version ;
 version
---------
 3.24.89
(1 wiersz)

acn=>
```

W przypadku rozbieżności należy upewnić się, która wersja aplikacji powinna zostać zainstalowana. Jeśli poprawna jest wersja w bazie danych, należy pobrać z Nexusa odpowiednią wersję plików EAR.
W przypadku, gdy konieczna jest zmiana wersji aplikacji w bazie danych należy wykonać skrypt:

```
BEGIN;
update master.system_version set version = '$VERSION';
COMMIT;
```

gdzie $VERSION to numer wersji.

Ruch pomiędzy serwerami

Do prawidłowego działania aplikacji potrzebne jest przepuszczenie ruchu pomiędzy serwerami aplikacyjnymi a serwerem bazy danych oraz pomiędzy serwerami aplikacyjnymi a serwerem HTTP. Szczegółowe założenia dotyczące ruchu opisuje dokument ITS (Projekt Infrastruktury Techniczno-Systemowej).

Klucze RSA

Do prawidłowego działania narzędzia clusterManager niezbędna jest możliwość logowania do serwera za pomocą klucza RSA. ClusterManager za pomocą narzędzia rsync przenosi spakowany klaster na serwery aplikacyjne oraz pliki statyczne na serwery HTTP. W przypadku używania clusterManagera na jednym z serwerów aplikacyjnych również należy skonfigurować logowanie po kluczu RSA (klaster jest przesyłany między serwerami).

Konfiguracja serwera DNS

Dla ustalonych domen aplikacji należy dodać wpis w konfiguracji serwera DNS. Dodany zapis powinien kierować bazową domenę z pliku konfiguracyjnego Apache'a `vhost.conf`, czyli `ServerName`, z uwzględnieniem wszystkich poddomen, na adres VIP'a LB dla danego środowiska. W przypadku środowiska DEV ta konfiguracja powinna kierować: `*projectcode-dev.xyz.pl` na adres IP `10.2.xx.yy`. Dozwolone jest również zdefiniowanie wpisów dla każdego aliasu z osobna, zdefiniowanego w konfiguracji Apache'a (`ServerAlias` w konfiguracji VirtualHost w pliku `vhosts.conf`) na adres serwera HTTP.

Instalacja aplikacji

Ten rozdział przedstawia aktualny, manualny sposób instalacji aplikacji na serwery. Po skonfigurowaniu narzędzia CI proces budowania i deploy aplikacji będzie odbywał się automatycznie, ale z wykorzystaniem obecnych narzędzi.

ClusterManager

Cluster Manager to narzędzie umożliwiające:

- przygotowanie archiwów instalacyjnych dla klastra wg. profilu osobno dla każdego węzła klastra,
- zarządzanie (`stop/start/kill/9kill`) elementami klastra z poziomu węzła za pomocą generowanego skryptu `NAZWAKLASTRA-admin.sh`,
- instalacja/aktualizacja aplikacji i jej konfiguracji (`deploy, cfe`).

Narzędzie jest udostępnione w repozytorium kodu aplikacji ACN o nazwie `wwwxyz-app`, w katalogu `projectcode_environments_manager/`.

Przygotowanie profilu

Zanim skorzystamy z cluster managera należy przygotować profil środowiska dla wdrażanej aplikacji. W tym celu tworzymy plik o nazwie `profile-acn-projectcode-[env].py` i uzupełniamy niezbędne szczegółowe informacje na temat przygotowywanego środowiska:

```
# -*- coding: utf -*-

"""
Profil dla środowiska DEV w projekcie projectcode.
Obowiązkowe elementy konfiguracji są oznaczone wykrzyknikiem na początku komentarza ( #! )
"""
import os
import sys
from python_libs import utils

USER=os.environ["USER"]
HOME=os.environ["HOME"]
JAVA_HOME=os.environ["JAVA_HOME"]
myIp = utils.get_my_external_ip()

class Config(object):

        projectName = "acn"    #! Nazwa projektu
        isGradle = 1           #! Flaga oznaczająca, że do budowania korzystamy z Gradle
```

```
        envName = "projectcode-dev" #! Nazwa środowiska
        projectDir = "{}/work/{}".format(HOME, projectName) #! Lokalny katalog projektu

        #! Workers prefix (in workers.properties)
        workerPrefix = "{envName}".format(envName=envName)

        # Nazwa ciasteczka na którym obowiązuje replikowana sesja
        # Wildfly zapewni obsługę sesji na danym ciasteczku
        sessionCookieName = "CMSSESSIONID"

        appVersion = "3.24.43" #! Wersja projektu

        #! Katalog zawierający szablony konfiguracyjne wildfly (w repozytorium projektu)
        customTemplates = "{projectName}/{envName}".format(projectName=projectName,
envName=envName)

        #! wrzucamy na zadalny serwer czy nie?
        developerMode = False

        applications = ["admin", "user", "crontasks", “acnlibs”] #! Lista aplikacji w rozumieniu
Wildfly

        # Namiary na truststore zawierający listę zaufanych certyfikatów
        # (ściezka względna wobec katalogu standalone)
        # sslKeyStorePath = "configuration/server.keystore"
        # sslKeyStorePassword = "ENCRYPTED"

        domain = "xyz.pl" #! bazowa/główna domena pod którą będzie dostępna aplikacja
        # Zmienna pomocnicza by skrócić zapisy w profilu,
        # wartosc zaczynajace sie od __ są ignorowane przez clusterManagera
        __serverName = "{}-{}.{}".format("{}", projectName, domain)

        # Lista definicji adresów http/domen poszczególnych aplikacji w rozumieniu Wildfly
        hosts = [
        {
        "name" : "user", # nazwa
        "aliases": ["user-{}.{}".format(envName, domain),
                        "*.user-{}.{}".format(envName, domain),
                        "xyz-user-{}.{}".format(envName, domain)
                        ], # lista domen pod którymi jest dostępna aplikacja
        "warmUpUrl": "/system_version" # url pod którym jest dokonywane “rozgrzanie” app
        },
        {
              "name" : "crontasks",
              "aliases": ["cron-{}.{}".format(envName, domain), __serverName.format("cron")],
        "warmUpUrl": "/system_version",
        },
        {
        "name": "admin",
        "aliases": ["admin-{}.{}".format(envName, domain), __serverName.format("admin")],
        "warmUpUrl": "/system_version"
        }
        ]

        isProductionEnv = False #! Czy profil dotyczy srodowisko produkcyjne (True lub False)
        clusterName = "{envName}-Cluster".format(envName=envName) #! Nazwa klastra
```

```
        wildflyUser = "projectcode_dev" # Użytkownik aplikacyjny OS (z którego odpalamy aplikację)

        # Katalog pomocniczych artefaktów np. wildfly-monitor
        productionDir = "/u01/app/{}".format(wildflyUser)

        #! Główny katalog zawierający wygenerowaną konfigurację serwerów klastra
        wildflyClusterBase = "/u01/app/{}/{}".format(wildflyUser, clusterName)

        #! Lokalny katalog z zbudowanymi aplikacjami do wrzucenia na serwer
        installableAppsDir = "{}/.m2/repository/{}".format(HOME, projectName)

        #! Katalog zawierający binarki serwera aplikacyjnego
        wildflyRoot = "/opt/jboss-eap-7.1"

        #! Ścieżka do binarek javy (wersja 1.8) z której jest odpalany serwer aplikacyjny
        jdkPath = "/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.141-2.b16.el7_4.x86_64/jre"

        # Ścieżka do binarek javy (wersja 1.8) z której jest odpalany monitoring serwera
        jdkPathMonitor = "/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.141-2.b16.el7_4.x86_64/jre"

        # Katalog z konfiguracją logowania dla aplikacji
        logbackConfigurationFile =
"{}/{}-{{node_name}}-server{{server_id}}/logback.xml".format(wildflyClusterBase, clusterName)

        #! Flaga czy należy kopiować statyczne zasoby na serwer (strony błędów)
        copyStaticContent = True

        # Adres mail do supportu aplikacji
        supportTeamEmailAddress = 'projectcode.dev.ac@xyz.pl'

        # Nazwa głównego logu aplikacji
        appLogFileName = 'acn.log'

        # Nazwa katalogu logu aplikacji
        appLogDir =
"{}/{}-{{node_name}}-server{{server_id}}/standalone/log".format(wildflyClusterBase, clusterName)

        # Ścieżka do pliku konfiguracji hazelcasta
        hazelcastConfigurationFile =
"{}/{}-{{node_name}}-server{{server_id}}/standalone/hazelcast.xml".format(wildflyClusterBase,
clusterName)


        #! Konfiguracja węzłów serwera aplikacyjnego
        nodes = [
        {
        "name": "osx02100", # Nazwa serwera
        "wildflyMonitoring": True, # Czy na serwerze jest włączony monitoring Wildflya
        "cronEntries": [ # wpisy do crontaba danego serwera
                    "1 0 * * *  jboss/wildfly-monitor/start.sh some.conf >/dev/null",
                    "* * * * *  {}/cron/crontask.sh -u /?action=t.a".format(wildflyClusterBase),
                   # Resztę cronów pominięto dla czytelności
        ],
        "servers": [ # Lista węzłów Wildfly na danym serwerze
                {
                "ip": "10.2.xx.yy", # adres IP
```



```
        "managementIp": "10.2.xx.yy", # adres IP dla monitoringu Wildfly
        "portBase": xx001, # bazowy port węzła - musi być zarezerwowane 100
                                # kolejnych portów dla danego węzła
        },
        {
        "ip": "10.2.xx.yy",
        "managementIp": "10.2.xx.yy",
        "portBase": xx001,
        }
]
},
{
"name": "osx02101",
"servers": [
        {
        "ip": "10.2.xx.yy",
        "portBase": xx001,
        },
        {
        "ip": "10.2.xx.yy",
        "portBase": xx101,
        }
]
}
]

# Konfiguracja namiarów na bazę danych
datasources = [
{
"jndiName" : "java:/jdbc/AcnDS", # Nazwa JNDI po której aplikacja szuka DS
"poolName" : "AcnDS",            # Nazwa puli połączeń
"enabled" : "true",              # Czy włączony DS
"newConnectionSql": "select 1",  # Zapytanie do nawiązania nowego połączenia
"checkValidConnectionSql": "select 1", # Zapytanie do sprawdzania czy połączenie
                                       # jest aktywne
"useJavaContext": "false",       # Czy prefiksować nazwę DS "java:"
"driver": "postgresql.jar",      # Namiary na sterownik bazy danych
"xaDataSourceClass" : "org.postgresql.xa.PGXADataSource", # Klasa sterownika do
                                                          # transakcji rozproszonych
"xaDatasourceProperties": {
             "serverName": "xxxx",   # Nazwa serwera bazy danych (BD)
             "portNumber": "xxxx",       # Port na którym BD nasłuchuje na połączenia
            "DatabaseName": "ac",        # Nazwa bazy danych (BD) aplikacji
            "User": "projectcode_ac_dev",      # Użytkownik BD
            "Password": "password",      # Hasło BD
},
"xaPool": {
            "minPoolSize": 2,            # Minimalny rozmiar puli połączeń do BD
            "maxPoolSize": 30,           # Maksymalny rozmiar puli połączeń do BD
            "prefill": "true"            # Czy wykonać prefill do minPoolSize
}
},
{
"jndiName" : "java:/jdbc/RuAcnDS", # Kopia DS wymagana przez aplikacje, powinna
"poolName" : "RuAcnDS",            #     wskazywać na te same namiary co AcnDS
# Ustawienia analogiczne jak w AcnDS - pominięte dla czytelności
```

```
},
{
"jndiName" : "java:/jdbc/AcnBigFilesDS", # DS do przechowywania dużych plików
"poolName" : "AcnBigFilesDS",
"xaDatasourceProperties": {
            "serverName": "xxx",
            "portNumber": "xxx",
            "DatabaseName": "acn-big-files",    # Osobna baza dla dużych plików
            "User": "projectcode_ac_dev",
            "Password": "password",
},
# Ustawienia analogiczne jak w AcnDS - pominięte dla czytelności
}
]

# Dodatkowe zasoby wymagane przez aplikację (pliki kopiowane do katalogu deployments)
otherDeployments = {
    "postgresql.jar": "acn/templates/lib/postgresql-9.1-901.jdbc4.jar"
}


# Dane uwierzytelniające do konsoli zarządzania wildfly
mgmtRealms = {
    "name" : "admin123",
    "password": "admin123"
}

# Konfiguracja hazelcasta
hazelcast = {
    "interface" : "{{node_name}}",
    "port" : str(nodes[0]["servers"][0]["portBase"]+90),
    "members" : [
"osx02100",
"osx02101"
    ],
    "ssl" : "false",
    "poolSize" : "16"
}
# Moduły, które mają być włączone
modules = {}
# Flagi dla środowiska konfigurujące serię mechanizmów aplikacji
properties = {
    "timeService.url": "time-{}".format(domain),
    "masterDomain": "user-{}.{}".format(envName, domain),
    "oneweb.console.server": "projectcodedev",
    "oneweb.i18n.cache": "false"
    # Resztę flag pominięto dla czytelności
}
```

Rys. 9 Profil środowiska `projectcode-dev`.

Gdy profil będzie gotowy możemy uruchomić skrypt `clusterManager.py`. Uruchamiamy go wpisując w wiersz poleceń:

```
./clusterManager.py -o [nazwa_operacji] -c [nazwa_pliku_z_profilem]
```


, gdzie:

- o oznacza operację do wykonania (cfe, deploy),
- c oznacza profil, z którego clusterManager będzie korzystać w trakcie działania.

Polecenie cfe generuje konfigurację środowiska i kopiuje ją na serwery aplikacyjne podane w profilu. CFE dla środowiska dev wykonujemy poprzez komendę

```
./clusterManager.py -o cfe -c profile-acn-projectcode-dev.py
```

Poprawnie zakończone wywołanie clusterManager'a powinno wyglądać następująco:

```
===> Creating full environment from scratch
===> PROGRESS: Preparing files with configuration.
===>  installation instruction for projectcode-dev-Cluster
Generated on Wed Apr 18 12:11:46 2018 by user
ssh HOST
su - projectcode_dev

*********************************************************
******* On APPLICATION servers: ['osx02100', 'osx02101'] *******
cd /u01/app/projectcode_dev
cp ~user/projectcode-dev/projectcode-dev-Cluster-dist.tar
~user/projectcode-dev/projectcode-dev-Cluster-install.py .
./projectcode-dev-Cluster-install.py install

*********************************************************
******* On HTTP servers: [] *******
* Create production/bin
mkdir -p ./production/bin

===> PROGRESS: Preparing archive with configuration.
===> PROGRESS: Uploading files to server.
CFE - success
```

Rys. 10 Rezultat wykonania cfe.

W rezultacie działania znajdziemy instrukcje co dalej należy zrobić, czyli:

- łączymy się poprzez ssh na serwery aplikacyjne:

```
ssh [nazwa_serwera_lub_adres_ip]
```

- logujemy się na użytkownika aplikacyjnego:

```
su - [nazwa_użytkownika]
```

- wklejamy polecenie będące rezultatem wykonania cfe z clusterManager'a:

```
cp ~user/projectcode-dev/projectcode-dev-Cluster-dist.tar
```



```
~user/projectcode-dev/projectcode-dev-Cluster-install.py .
```

- wykonujemy skrypt instalacyjny aktualizujący konfigurację środowiska:

```
./projectcode-dev-Cluster-install.py install
```

```
===> ProductionDir: /u01/app/projectcode_dev
 installed bin
 installed www
 installed apps
 installed log
 installed cron tasks
 installed wildfly-monitor
 wildlfy monitor crontab entry ok
```

Rys. 11 Rezultat wykonania poleceń wygenerowanych przez cfe - kopiowanie konfiguracji środowiska wraz z uruchomieniem skryptu instalacyjnego.

W

Do poprawnego działania aplikacji niezbędny jest sterownik postgresql.jar. W celu zainstalowania go należy przekopiować plik postgresql-42.2.2.jre7.jar z lokalizacji projectcode_environments_manager/jboss7/acn/templates/lib/postgresql-42.2.2.jre7.jar do utworzonego folderu (przez cfe) z aplikacją: production/apps/postgresql.jar.

Uzupełnienie bazy danych

Przed uruchomieniem aplikacji należy uzupełnić bazę danych poprzez wykonanie skryptu. Skrypt znajduje się w aplikacji SharePoint w lokalizacji *projectcode/Dokumenty Operacyjne/Dokumentacje/Binaria/skrypty sql/acn.* Skrypt został spakowany i podzielony na części. W celu połączenia części w paczkę należy wykonać polecenie:

```
cat create_projectcode_acn_db.z01 create_projectcode_acn_db.z02
create_projectcode_acn_db.z03 create_projectcode_acn_db.z04 [...]
create_projectcode_acn_db.z26 create_projectcode_acn_db.zip >
create_projectcode_acn_joined.zip
```

Następnie należy rozpakować skrypt

```
unzip create_projectcode_acn_joined.zip
```

Rozpakowany skrypt *create_projectcode_acn.sql* należy wgrać do bazy danych:


```
psql -h [host_bazy_danych] -U [user_bazy_danych] -p [port_bazy_danych]
-d [baza_danych] -f create_projectcode_acn.sql
```

Uzupełnienie konfiguracji środowiska w bazie danych

Zmienne Common Properties

W schemacie *common_xyz* znajdują się dwie tabele odpowiedzialne na zmienne współdzielone, czyli wspólne wartości wykorzystywane przez wszystkie serwisy udostępnione w ramach jednego klienta. Tabela *common_properties_type* jest tabelą słownikową przechowującą nazwy i identyfikatory zmiennych, a *common_properties* ich wartość. Poniżej znajduje się lista kluczowych zmiennych wraz z ich domyślnymi wartościami, które powinny znaleźć się w bazie danych:

client_code	xyz
is_multilanguage_enabled	true
notification_from_email	<adres email z którego mają przychodzić powiadomienia>
file_manager_compression_type_id	1
show_file_download_statistics_in_file_details	true
timezone	Europe/Warsaw

Zmienne Site Properties

W schemacie *site_xyz_1401* znajdują się dwie tabele odpowiedzialne za zmienne dla portalu. Tabela *site_properties_type* jest tabelą słownikową przechowującą nazwy i identyfikatory zmiennych, a *site_properties* ich wartości. Poniżej znajduje się lista kluczowych zmiennych wraz z ich domyślnymi wartościami, które powinny znaleźć się w bazie danych:

site_name	xyz
homepage_url	/aaaaaaaaaa
search_page_url	/bbbbbbbbbb
page_not_found_url	/404

access_forbidden_url	/500
site_search_file_extensions	pdf
login_required_url	/trwa-aktualizacja-systemu
xyz_job_offers_agreement_question_id	rasqu000000000000011
af_client_base_url	xyz
news_details_default_url	/grupa-xyz/o-nas/aktualnosci/szczegoly

Dodatkowe zmienne

Poza zmiennymi znajdującymi się w tabelach z przyrostkiem properties, istnieją również dedykowane encje przechowujące konfiguracje dla danych funkcjonalności.

Do jednej z nich należy tabela *sso_token* w schemacie *master*, która przechowuje tokeny SSO wymagane do integracji z ActiveForms. Jej format jest następujący:

client_account_id	**api_token**	**secret_api_token**
14	<klucz API w ActiveForms>	<sekretny klucz API w ActiveForms>

Tabela *system_version* w schemacie *master* przechowuje aktualny numer wersji aplikacji, który powinien być zgodny z wersją znajdującą się w pliku *build.gradle* w repozytorium aplikacji. Jej format jest następujący:

id	**version**
1	<wersja aplikacji>

Uruchomienie aplikacji

Budowanie aplikacji

Aplikację budujemy z głównego katalogu projektu wykonując skrypt:

```
./build.sh
```

Poprawne wyjście skryptu zakończone jest liniami:


```
:war-user:generatePomFileForMavenJavaPublication
:war-user:jar
:war-user:publishMavenJavaPublicationToMavenLocal
:war-user:publishToMavenLocal

BUILD SUCCESSFUL

Total time: 4 mins 7.811 secs
```

Skrypt buduje odpowiednie archiwa ear/war i publikuje je w lokalnym mavenie.

Uruchomienie aplikacji ze zbudowanych paczek

W przypadku, gdy uruchamiamy aplikację z dostarczonych plików ear (bez budowania aplikacji w repozytorium) należy w profilu uzupełnić ścieżkę do tych plików. Pliki należy umieścić w wybranej lokalizacji w folderze “ears”. W profilu należy przypisać zmiennej *installableAppsDir* wartość wybranej lokalizacji.

Przeniesienie aplikacji na serwery aplikacyjne

W celu skopiowania zbudowanej aplikacji na serwery, wykonujemy skrypt clusterManager z operacją deploy:

```
./clusterManager.py -o deploy -c profile-acn-projectcode-dev.py
```

Polecenie przenosi paczki do wskazanego w profilu katalogu miejsca instalacji aplikacji. Prawidłowo zakończone wywołanie clusterManager’a z poleceniem `deploy` powinno wyglądać następująco:

```
===> PROGRESS: Creating directories.
===> PROGRESS: Locally copying ears.
===> PROGRESS: Generating start scripts.
===> PROGRESS: Copying static resources.
===> PROGRESS: Locally copying other resources.
===> PROGRESS: Uploading postgresql.jar to APP osx02100.

ssh HOST
su - projectcode_dev

**********************************************************
******* On APPLICATION servers: ['osx02100', 'osx02101'] *******
* Stop application
/u01/app/projectcode_dev/production/bin/projectcode-dev-Cluster-admin.py
stop
```



```
* Copy current admin.sh
cp ~user/projectcode-dev/projectcode-dev-Cluster-admin.py
/u01/app/projectcode_dev/production/bin/

* Deploy applications
/u01/app/projectcode_dev/production/bin/projectcode-dev-Cluster-admin.py
app_deploy

* Start applications
/u01/app/projectcode_dev/production/bin/projectcode-dev-Cluster-admin.py
start

************************************************
******* On HTTP servers: [] *******
* Copy current admin script
cp ~user/projectcode-dev-Cluster-admin.py
/u01/app/projectcode_dev/production/bin/

* Deploy applications
/u01/app/projectcode_dev/production/bin/projectcode-dev-Cluster-admin.py
http_deploy

 DEPLOY - success
```

Rys. 12 Rezultat wykonania operacji deploy z clusterManager'a.

W rezultacie znajdziemy instrukcje, co dalej należy zrobić, czyli:

- łączymy się poprzez ssh na serwery aplikacyjne:

```
ssh nazwa_serwera_lub_adres_ip
```

- logujemy się na użytkownika aplikacyjnego:

```
su - nazwa_użytkownika
```

- jeśli aktualizujemy wersję zatrzymujemy węzły klastra działające na aktualnym serwerze

```
./projectcode-dev-Cluster-admin.py stop
```

- wklejamy polecenie będące rezultatem wykonania deploy z clusterManager'a:

```
cp ~user/projectcode-dev-Cluster-admin.py path_to_app/production/bin
```

- wykonujemy deploy aplikacji

```
./projectcode-dev-Cluster-admin.py app_deploy
```



```
 Deployed ear-admin-3.24.43.ear
 Deployed ear-user-3.24.43.ear
 Deployed ear-crontasks-3.24.43.ear
 Deployed postgresql.jar
 Deployed static pages
```

- uruchamiamy węzły klastra działające na aktualnym serwerze

```
./projectcode-dev-Cluster-admin.py start
```

Dla każdego węzła zobaczymy listę komunikatów:

```
 starting server projectcode-dev-Cluster-osx02100-server1
 Elasped seconds: 63
 Server is started

===> Warming up server
 Warming up user succeeded
 Warming up crontasks succeeded
 Warming up admin succeeded
===> executing post_start.sh
```

Logi aplikacji dostępne są w lokalizacji:

```
[katalog_z_aplikacją]/production/logs/[nazwa_węzła_klastra]/
```

Aplikację obsługujemy skryptem `projectcode-dev-admin-Cluster.py` znajdującym się w katalogu `[katalog_aplikacji]/production/bin`. Możemy go wykonać z parametrem:

- **status** - pokazuje aktualny status działania węzłów aplikacji na tym serwerze, wyświetla nazwy węzłów w klastrze,
- **start** - startuje wszystkie węzły na tym serwerze,
- **stop** - zatrzymuje wszystkie węzły na tym serwerze w sposób łagodny, zezwalając na dokończenie obsługi przetwarzanych żądań,
- **kill** - zatrzymuje natychmiast wszystkie węzły aplikacji na tym serwerze (TERM),
- **9kill** - zatrzymuje natychmiast wszystkie węzły aplikacji na tym serwerze (SIGKILL),
- **ports** - wyświetla listę portów serwera.

Jako dodatkowy parametr do operacji `start`, `stop` i `kill` można dodać numer węzła, wtedy podana operacja będzie dotyczyć tylko wskazanego węzła (numer węzła liczony od jedynki).

