# Jak korzystać z Grida (Pierwsze kroki)

1. By zacząć pracę z Gridem użytkownik musi posiadać własny certyfikat X.509 oraz konto na komputerze, którym zainstalowane jest oprogramowanie WLCG/EGEE User Interface np. serwis CERN LXPLUS może być wykorzystany jako gLite User Interface.
2. Użytkownik musi być zapisany do co najmniej jednej wirtualnej organizacji (VO). Poniżej znajduje się lista wirtualnych organizacji związanych z eksperymentem LHC:
   - ALICE
   - ATLAS
   - CMS
   - DTEAM
   - LHCB
   - SixTrack

   Aby uzyskać przynależność do jakiejś VO, należy wypełnić formularz z wnioskiem o przynależność do danej organizacji. Przykładowo dla VO ALICE formularz taki można znaleźć na stronie: *https://lcg-voms.cern.ch:8443/vo/alice/vomrs*

   Należy pamiętać, by mieć możliwość wypełnienia takiego formularza należy wcześniej załadować swój podpisany certyfikat do przeglądarki. Ponieważ przeglądarki internetowe używają innego formatu certyfikatów (PKCS12) niż certyfikaty gridowe (PEM) to istnieje konieczność przekonwertowania certyfikatu PEM na certyfikat obsługiwany przez przeglądarki. Przekonwertowania można dokonać na dowolnym komputerze, który posiada zainstalowany pakiet *openssl*, należy z konsoli wydać polecenie:

```
openssl pkcs12 -export -inkey userkey.pem -in
usercert.pem -out my_cert.p12 -name "My certificate"
```

   gdzie:
   - *userkey.pem* – jest ścieżką dostępu do naszego pliku *userkey.pem*
   - *usercert.pem* – jest ścieżką dostępu do naszego pliku *usercert.pem*

- *my_cert.p12* – jest ścieżką do pliku wyjściowego w formacie PKCS12, który ma zostać utworzony
- *My certificate* – jest nazwą naszego certyfikatu

Poniżej znajduje się opis ładowania certyfikatu dla dwóch podstawowych rodzajów przeglądarek:

- **Firefox** (wersja 3.5.5)
  - Uruchom przeglądarkę Firefox
  - Z górnego menu wybierz 'Narzędzia' -> 'Opcje'
  - Wybierz zakładkę 'Zaawansowane' następnie zakładkę 'Szyfrowanie'
  - Kliknij przycisk 'Wyświetl certyfikaty' po czym przejdź do zakładki 'Certyfikaty użytkownika'
  - Kliknij przycisk 'Importuj…' po czym wskaż ścieżkę dostępu do stworzonego pliku *my_cert.p12*
- **Internet Explorer** (wersja 7)
  - Uruchom przeglądarkę Internet Explorer
  - Z górnego menu wybierz 'Narzędzia' -> 'Opcje internetowe'
  - Wybierz zakładkę 'Zawartość' następnie kliknij przycisk 'Certyfikaty'
  - Przejdź do zakładki 'Osobisty', po czym kliknij przycisk 'Importuj' i wskaż ścieżkę dostępu do stworzonego pliku *my_cert.p12*

3. Należy zalogować się na komputer z zainstalowanym oprogramowaniem WLCG/EGEE User Interface, np. do serwisu CERN LXPLUS. Na komputerze dostępowym należy stworzyć katalog *.globus* w katalogu domowym (*$HOME*). Następnie należy skopiować swój certyfikat do tego katalogu (przenieść pliki *usercert.pem* i *userkey.pem* do katalogu *$HOME/.globus*). Zważywszy na kwestie bezpieczeństwa dla powyższych plików należy ustawić następujące prawa dostępu:

```
chmod 0444 ~/.globus/usercert.pem
chmod 0400 ~/.globus/userkey.pem
```

Po wydaniu z konsoli polecenia *ls -l $HOME/.globus/* powinniśmy otrzymać następujący wynik:

```
total 1
-r--r--r--  1 doe xy 4989 Nov  9 00:54 usercert.pem
-r--------  1 doe xy  963 Nov  9 00:54 userkey.pem
```

Otrzymanie takiego wyniku świadczy o prawidłowym i bezpiecznym przechowywaniu certyfikatu.

4. Następnym krokiem jest ustawienie zmiennych środowiskowych. By tego dokonać należy uruchomić jedną z poniższych komend w zależności od używanej konsoli:

   - Dla użytkowników konsoli *Bourne Shell:*

     source /afs/cern.ch/project/gd/LCG-share/sl4/etc/profile.d/grid_env.sh

   - Dla użytkowników konsoli *C shell:*

     source /afs/cern.ch/project/gd/LCG-share/sl4/etc/profile.d/grid_env.csh

   W plikach *grid_env.sh* i *grid_env.csh* znajdują się wszystkie potrzebne zmienne środowiskowe, które zostaną automatycznie załadowane po uruchomieniu jednego z powyższych skryptów. Skrypty te dadzą nam zawsze dostęp do najnowszego oprogramowania UI dostępnego na LXPLUS.

5. Należy sprawdzić, czy nasz certyfikat jest poprawnie zainstalowany. W tym celu z linii komend należy wydać polecenie:

   ```
   grid-cert-info
   ```

   W celu sprawdzenia ważności naszego certyfikatu należy wydać polecenie:

   ```
   grid-cert-info –enddate
   ```

6. Certyfikatu użytkownika, którego klucz prywatny jest zabezpieczony hasłem, używa się do wygenerowania i podpisania tymczasowego certyfikatu zwanego proxy. Głównym celem proxy jest umożliwienie identyfikacji użytkownika w gridzie.

Warto podkreślić, że w celach bezpieczeństwa proxy ma krótką żywotność, np. 12h. Certyfikat proxy tworzymy poleceniem:

```
voms-proxy-init –voms <vo>
```

Jako <vo> podajemy nazwę wirtualnej organizacji, do której przynależymy np.

```
voms-proxy-init –voms alice
```

Wynik powinien być podobny do poniższego:

```
Your identity: /C=PL/O=GRID/O=Warsaw University of Technology Faculty of
Physics/CN=John Doe
Creating temporary proxy
.................................................................. Done
Contacting  voms.cern.ch:15000
[/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch] "alice" Done
Creating proxy ................................................... Done
Your proxy is valid until Fri Dec  4 10:30:57 2009
```

Nasz certyfikat proxy został utworzony standardowo na 12h. Powyższą komendą oprócz utworzenia proxy, można także przedłużyć już istniejące. W celu stworzenia proxy na określony przez użytkownika czas należy posłużyć się komendą:

```
voms-proxy-init –voms alice –valid H:M
```

gdzie jako H:M należy podać czas na jaki chcemy utworzyć proxy, np. polecenie

```
voms-proxy-init –voms alice –valid 0:10
```

tworzy proxy na 10 minut. Należy pamiętać, że im dłuższa żywotność proxy tym większe ryzyko przechwycenia przez niepowołaną osobę. W celach bezpieczeństwa należy niszczyć proxy po wykonanych obliczeniach. Służy do tego polecenie:

```
voms-proxy-destroy
```

W celu sprawdzenia informacji o istniejącym już proxy takich jak czas jego życia należy posłużyć się komendą:

```
voms-proxy-info
```

Warto podkreślić, że jeśli okres życia proxy skończy się, kiedy mamy jeszcze „puszczone” zadanie w gridzie, to zadanie takie nie zostanie wykonane.

7. By przesłać zadanie na grid należy przygotować plik z opisem zadania, który zawiera informacje dla Resource Brokera typu:

    - Typ naszego zadania, czy jest ono szeregowe czy równoległe
    - Ilość procesorów potrzebnych do wykonania naszego zadania
    - Pliki wejściowe i wyjściowe
    - Gdzie znajduje się plik wykonywalny

Do opisu zadania używa się specjalnego języka zwanego JDL (Job Description Language). Warto podkreślić, że w pliku JDL można jawnie zdefiniować CE (klaster) na którym ma zostać wykonane nasze zadanie. W celu uruchomienia prostego zadania w gridzie, które zwracać będzie nazwę komputera, plik JDL powinien wyglądać następująco:

```
JobType = "Normal";
Executable = "/bin/hostname";
StdOutput = "wynik.out";
StdError = "bledy.err";
OutputSandbox = {"wynik.out","bledy.err"};
```

Powyższe zadanie w gridzie wywoła komendę */bin/hostname,* poczym wynik tej komendy zostanie zapisany do pliku *wynik.out,* jeśli wystąpi błąd podczas wykonywania zadania, informacje o tym błędzie zostaną zapisane w pliku *bledy.err.* Aby uruchomić zadanie wydajemy komendę:

```
glite-wms-job-submit –a plik_jdl
```

gdzie *plik_jdl* jest plikiem o rozszerzeniu **.jdl,* zawierającym opis zadania w gridzie.

Jako wynik powyższej komendy powinniśmy otrzymać informację podobną do poniższej:

Connecting to the service https://wms215.cern.ch:7443/glite_wms_wmproxy_server

========== glite-wms-job-submit Success ==========

The job has been successfully submitted to the WMProxy

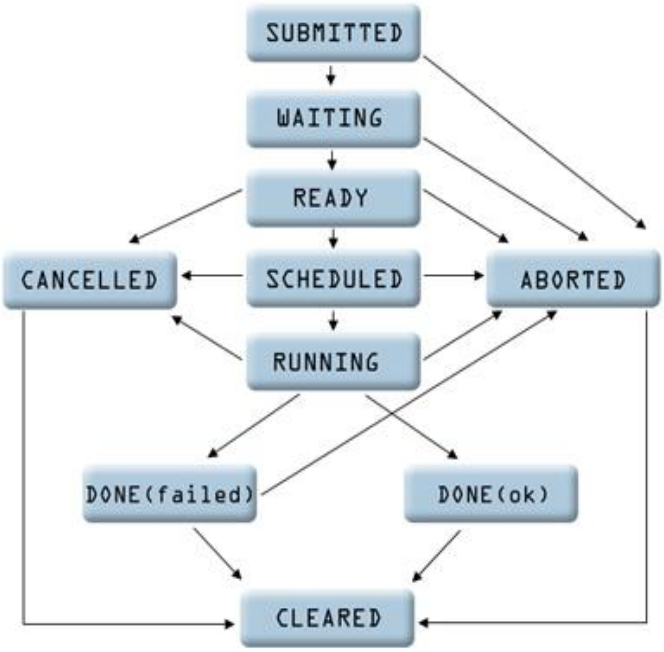
Your job identifier is:

https://wms215.cern.ch:9000/Ft9mvXIGaOFwulN9dspLtg

==========

Taka informacja świadczy o poprawnym dodaniu zadania do system kolejkowego w gridzie. Nasze zadanie otrzymało identyfikator, który należy zapisać ponieważ będzie on nam potrzebny do sprawdzania statusu naszego zadania, czy choćby do jego odebrania.

8. Prawidłowo dodane zadanie powinno znaleźć się w jednym z poniższych stanów:




- *SUBMITTED – oznacza, że zadanie zostało dodane przez użytkownika ale nie przeszło jeszcze walidacji.*
- *WAITING – zadanie przeszło walidację, w tym momencie następuje wybór przez Resource Brokera najbardziej optymalnego serwera na którym można wykonać nasze zadanie.*
- *READY – Computing Element (CE) dla naszego zadania został wybrany*
- *SCHEDULED – Zadanie czeka w lokalnym systemie kolejkowym na CE*
- *RUNNING – Nasze zadanie jest w trakcie wykonywania*
- *DONE (ok) – Zadanie zostało ukończone pomyślnie*
- *DONE (failed) – Zadanie zakończyło się błędem*
- *CLEARED – Wynik zakończonego zadania został przesłany do User Interface.*
- *CANCELLED – Zadanie zostało anulowane przez użytkownika*
  *ABORTED – Zadanie zostało przerwane przez oprogramowanie gridowe*

**Rysunek 6.** Etapy wykonywania zadania w gridzie

# Uruchamianie zadań równoległych

**Message Passing Interface (MPI)** – jest nazwą standardu biblioteki, przeznaczonej do przesyłania komunikatów wykorzystywanych w programowaniu równoległym. Ze standardem MPI można spotkać wszędzie tam, gdzie mamy do czynienia z systemami rozproszonymi (np. GRID). MPI udostępnia nam szeroką gamę funkcji dla różnych języków programowania (C/C++, Fortran). W celu uruchomienia zadania równoległego w gridzie należy przygotować cztery pliki:

- *mpi-start-wrapper.sh* – skrypt ustawiający odpowiednie zmienne środowiskowe do wykonywania zadań równoległych, za pomocą zmiennych środowiskowych dostarczonych przez administratora systemu. Określa pliki wykonywalne, argumenty oraz lokalizację skryptów *hooks.* Skrypt ma charakter wewnętrzny i nie potrzeba dokonywać w nim żadnych modyfikacji.
- *mpi-hooks.sh* – Istnieją dwa rodzaje skryptów tego typu *Pre-hook* i *post-hook*. *Pre-hook* jest wykonywany przed wykonaniem pliku wykonywalnego naszego zadania MPI. Przykładowo skrypt taki może posłużyć do kompilacji pliku wykonywalnego czy pobierania danych potrzebnych do wykonania naszego zadania. *Post-hook* wykonywany jest po uruchomieniu pliku wykonywalnego. Wykorzystywany jest zazwyczaj do analizy, czy do zapisania wyników w gridzie.
- *mpi-test.jdl* – plik w którym znajduje się opis naszego zadania, poniżej znajduje się przykład opisu zadania równoległego:

```
JobType         = "MPICH";
CPUNumber       = 16;
Executable      = "mpi-start-wrapper.sh";
Arguments       = "mpi-test OPENMPI";
StdOutput       = "mpi-test.out";
StdError        = "mpi-test.err";
InputSandbox    = {"mpi-start-wrapper.sh","mpi-
hooks.sh","mpi-test.c"};
OutputSandbox   = {"mpi-test.err","mpi-test.out"};
Requirements =
Member("MPI-START",
other.GlueHostApplicationSoftwareRunTimeEnvironment)
&& Member("OPENMPI",
other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Widać, że powyższa struktura pliku JDL niewiele różni się od struktury przedstawionej na stronie 19, która opisywała zadanie szeregowe. W tym przypadku opis naszego zadania jest powiększony o linię *CPUNumber = 16*, która musi być określona dla zadania równoległego. Atrybut ten określa ile procesorów będzie wykorzystanych do liczenia naszego zadania (16 w tym przykładzie). Kolejną ważną rzeczą jest określenie typu zadania (*JobType*), które dla zadań równoległych musi być *MPICH.*

- *mpi-test.c* – Plik ten powinien zawierać kod naszego programu przygotowany przy użyciu biblioteki MPI, w jednym z języków obsługiwanych przez tą bibliotekę.

# Przykładowe skrypty do wykonywania zadań równoległych

Wrapper skrypt (*mpi-start-wrapper.sh*) [6]

```
#!/bin/bash

MY_EXECUTABLE=`pwd`/$1
MPI_FLAVOR=$2
MPI_FLAVOR_LOWER=`echo $MPI_FLAVOR | tr '[:upper:]' '[:lower:]'`

eval MPI_PATH=`printenv MPI_${MPI_FLAVOR}_PATH`
eval I2G_${MPI_FLAVOR}_PREFIX=$MPI_PATH
export I2G_${MPI_FLAVOR}_PREFIX

touch $MY_EXECUTABLE

export I2G_MPI_APPLICATION=$MY_EXECUTABLE
export I2G_MPI_APPLICATION_ARGS=
export I2G_MPI_TYPE=$MPI_FLAVOR_LOWER
export I2G_MPI_PRE_RUN_HOOK=mpi-hooks.sh
export I2G_MPI_POST_RUN_HOOK=mpi-hooks.sh
export I2G_MPI_START_VERBOSE=1

$I2G_MPI_START
```

Hooks skrypt (*mpi-hooks.sh*)

```
#!/bin/sh
pre_run_hook () {
  echo "Compiling ${I2G_MPI_APPLICATION}"
  cmd="mpicc ${MPI_MPICC_OPTS} -o ${I2G_MPI_APPLICATION}
${I2G_MPI_APPLICATION}.c"
  echo $cmd
  $cmd
  if [ ! $? -eq 0 ]; then
    echo "Error compiling program.  Exiting..."
    exit 1
  fi
  echo "Successfully compiled ${I2G_MPI_APPLICATION}"
  return 0
}
post_run_hook () {
  echo "Executing post hook."
  echo "Finished the post hook."
  return 0
}
```