Everest Documentation

Выпуск 1.0-SNAPSHOT

Oleg Sukhoroslov

30 November 2012

Оглавление

Установка

1.1 Системные требования

Поддерживаются Windows и Linux-совместимые ОС.

Требуется Java SE 6 или 7.

1.2 Установка

Загрузите дистрибутив Everest и распакуйте в любую директорию, которую далее будем называть EVEREST_HOME.

Конфигурация

Конфигурационные файлы контейнера находятся в директории EVEREST_HOME/conf.

Конфигурация контейнера зависит от того, в каком режиме предполагается запускать контейнер:

- Публичный режим (public mode) – доступ к сервисам по протоколу HTTP без аутентификации клиентов;
- Защищенный режим (secure mode) – доступ к сервисам по протоколу HTTPS с аутентификацией клиентов.

При первоначальном знакомстве и отладке сервисов на локальной машине рекомендуется использовать публичный режим.

Защищенный режим позволяет ограничить круг клиентов, которые имеют доступ к сервисам, и рекомендуется при развертывании сервисов в режиме production.

2.1 Конфигурация веб-сервера

Для работы контейнера используется встроенный веб-сервер Jetty. Конфигурация Jetty осуществляется с помощью следующих файлов:

- EVEREST_HOME/conf/jetty.xml (для публичного режима)
- EVEREST_HOME/conf/jetty-ssl.xml (для защищенного режима)

Для публичного режима можно использовать конфигурацию по-умолчанию, входящую в дистрибутив. В этом случае веб-сервер будет использовать порт 8080.

Описание конфигурационных параметров Jetty можно найти здесь: http://wiki.eclipse.org/Jetty/Reference/jetty.xml

2.2 Конфигурация контейнера

Конфигурационные параметры непосредственно контейнера устанавливаются в файле EVEREST_HOME/conf/everest.conf. Данный файл использует формат JSON.

Поддерживаются следующие конфигурационные параметры контейнера:

- pool-size - размер пула потоков, осуществляющих обработку запросов к сервисам (заданий). При установке значения 0 размер пула равен числу процессорных ядер. [Необязательный параметр, значение по-умолчанию: 0]

2.3 Конфигурация безопасности

Данный раздел относится только к защищенному режиму. Пропустите его при использовании публичного режима (перейти к *Запуск и остановка*).

2.3.1 Установка сертификата сервера

В защищенном режиме доступ к сервисам предоставляется с помощью протокола HTTPS. Для этого необходимо снабдить веб-сервер контейнера SSL-сертификатом. Возможны два способа получения данного сертификата:

1. Создание самоподписанного (self-signed) сертификата сервера;
2. Получение сертификата сервера через некоторый удостоверяющий центр (Certificate Authority).

Первый способ является более простым с точки зрения установки, а второй способ – более безопасным с точки зрения клиентов. Далее рассмотрим подробнее первый способ.

Для создания самоподписанного сертификата сервера можно использовать команду

```
$ keytool -keystore keystore -alias everest -genkey -keyalg RSA -validity 365
```

При этом необходимо ввести информацию о сертификате и пароли, защищающие доступ к файлу с сертификатом и секретному ключу. Единственной обязательной для ввода информацией о сертификате является поле “first and last name”, куда требуется ввести полное имя хоста, на котором будет запущен контейнер. Опция “validity” устанавливает срок действия сертификата в днях. По истечении срока действия, необходимо создать новый сертификат и заменить им старый.

Пример запуска команды keytool:

```
$ keytool -keystore keystore -alias everest -genkey -keyalg RSA -validity 365
Enter keystore password:  password
What is your first and last name?
  [Unknown]:  dcs.isa.ru
What is the name of your organizational unit?
```

```
  [Unknown]:  CGT&DC
What is the name of your organization?
  [Unknown]:  ISA RAS
What is the name of your City or Locality?
  [Unknown]:  Moscow
What is the name of your State or Province?
  [Unknown]:  Moscow
What is the two-letter country code for this unit?
  [Unknown]:  RU
Is CN=dcs.isa.ru, OU=CGT&DC, O=ISA RAS, L=Moscow, ST=Moscow, C=RU correct?
  [no]:  yes
Enter key password for <everest>
        (RETURN if same as keystore password):  password
```

После успешного выполнения команды keytool необходимо скопировать файл keystore, содержащий созданный сертификат, в директорию EVEREST_HOME/conf.

2.3.2 Конфигурация веб-сервера в защищенном режиме

После того, как keystore-файл с сертификатом сервера размещен в директории EVEREST_HOME/conf, необходимо настроить веб-сервер для использования сертификата.

Для этого окройте файл EVEREST_HOME/conf/jetty-ssl.xml, найдите секцию “Set connectors” и отредактируйте ее следующим образом:

```
<Call name="addConnector">
  <Arg>
    <New class="org.eclipse.jetty.server.ssl.SslSelectChannelConnector">
      ...
      <Set name="Keystore">conf/keystore</Set>
      <Set name="Password">пароль keystore-файла (см. предыдущий раздел)</Set>
      <Set name="KeyPassword">пароль ключа (см. предыдущий раздел)</Set>
      <Set name="truststore">conf/keystore</Set>
      <Set name="trustPassword">пароль keystore-файла</Set>
      <Set name="NeedClientAuth">false</Set>
      <Set name="WantClientAuth">true</Set>
    </New>
  </Arg>
</Call>
```

2.3.3 Настройка аутентификации

Основным поддерживаемым методом аутентификации клиентов является аутентификация при помощи сертификатов формата X.509, подписанных доверяемым удостоверяющим центром. Для использования данного метода требуется

добавить корневые сертификаты доверяемых удостоверяющих центров в файл EVEREST_HOME/conf/keystore с помощью команды

```
$ keytool -keystore EVEREST_HOME/conf/keystore -import -alias myCA -file myCA.pem \
-trustcacerts
```

Контейнер также поддерживает дополнительный метод аутентификации при помощи публичных провайдеров учетных записей (через Loginza.API). Данный метод доступен только при доступе через веб-интерфейс при помощи браузера.

2.3.4 Настройка авторизации

Настройка авторизации заключается в определении двух списков клиентов:

- Клиенты, которым разрешен доступ (т.н. allow-список);
- Клиенты, которым запрещен доступ (т.н. deny-список).

Данные списки могут быть определены как на уровне контейнера, так и на уровне отдельного сервиса. В первом случае определяются глобальные списки, которые регламентируют доступ ко всем сервисам внутри контейнера. Во втором случае определяются локальные списки доступа для конкретного сервиса.

Глобальные списки доступа должны размещаться в файлах

- EVEREST_HOME/conf/allow
- EVEREST_HOME/conf/deny

Локальные списки доступа к сервису должны размещаться в файлах

- EVEREST_HOME/services/SERVICE_ID/allow
- EVEREST_HOME/services/SERVICE_ID/deny

Обратите внимание, что если ни глобальные, ни локальные списки не определены, доступ к сервисам не имеет ни один клиент.

Каждый из списков оформляется в виде файла, содержащего уникальные идентификаторы клиентов, по одному на строку. В качестве идентификаторов клиентов могут использоваться:

- Значение поля Distinguished Name (DN) сертификата клиента (используется при аутентификации с помощью цифрового сертификата);
- Идентификатор, полученный от публичного провайдера учетных записей (используется при аутентификации через сервис Loginza).

Идентификатор конкретного пользователя можно определить с помощью веб-интерфейса контейнера. После аутентификации искомый идентификатор отображается в правой части верхнего меню.

Также допускается использование символа “*” для обозначения клиента с любым идентификатором. Наличие данного символа в allow-списке означает, что доступ

разрешен любому клиенту. Наличие данного символа в deny-списке означает, что доступ запрещен всем клиентам.

Допускается создание только allow- или deny-списка, или же их полное отсутствие (запрещен доступ всем клиентам).

Успешная авторизация клиента происходит тогда и только тогда, когда выполняются следующие условия:

1. идентификатор клиента (или “*”) присутствует в глобальном ИЛИ локальном allow-списках;
2. идентификатор клиента (или “*”) отсутствует в глобальном И локальном deny-списках.

2.4 Дополнительная конфигурация

2.4.1 Настройка nginx

В случае, если перед контейнером будет использоваться nginx, конфигурация последнего должна выглядеть примерно так:

```
server {
    listen 80;
    server_name somehost.com;

    access_log /var/log/nginx/localhost.access.log;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header X-Real-IP  $remote_addr;
        proxy_set_header Host $host;
    }
}
```

Обратите внимание на использование директивы “proxy_set_header Host $host”. Она необходима для генерации контейнером корректных URI.

Дополнительную информацию по теме можно найти здесь и здесь.

Запуск и остановка

3.1 Запуск

Запуск в публичном режиме:

- Linux: EVEREST_HOME/bin/start.sh
- Windows: EVEREST_HOME/bin/start.bat

Запуск в защищенном режиме:

- Linux: EVEREST_HOME/bin/start.sh ssl
- Windows: EVEREST_HOME/bin/start.bat ssl

3.2 Проверка

Проверить, что контейнер запущен, можно открыв в браузере его веб-интерфейс

- В публичном режиме: http://HOST:PORT/ (номер порта по-умолчанию - 8080, см. EVEREST_HOME/conf/jetty.xml)
- В защищенном режиме: https://HOST:PORT/ (номер порта по-умолчанию - 8443, см. EVEREST_HOME/conf/jetty-ssl.xml)

Для диагностирования возможных проблем рекомендуется заглянуть в лог-файл контейнера: EVEREST_HOME/log/server.log

3.3 Остановка

- Linux: EVEREST_HOME/bin/stop.sh
- Windows: Ctrl+C

Разработка сервисов

4.1 Общие положения

Процесс разработки сервиса в общем случае состоит из следующих этапов:

1. Создание реализации сервиса;
2. Подготовка конфигурационного файла (дескриптора) сервиса;
3. Размещение сервиса в контейнере.

Контейнер позволяет легко, во многих случаях без написания программного кода, преобразовать в сервис широкий спектр существующих приложений. Поэтому первый этап разработки зачастую отсутствует.

4.2 Адаптеры

Для упрощения преобразования в сервисы существующих приложений используется механизм адаптеров. Адаптер осуществляет преобразование запроса к сервису в вызов внешнего приложения на том же сервере или запуск приложения в вычислительной инфраструктуре. При разработке сервиса необходимо выбрать один из поддерживаемых контейнером адаптеров.

В настоящее время реализовано четыре универсальных адаптера, поддерживающих интеграцию следующих типов приложений:

- приложение с интерфейсом командной строки (адаптер command, входит в состав дистрибутива контейнера);
- приложение на языке Java (адаптер java, входит в состав дистрибутива контейнера);
- грид-приложение, запускаемое в инфраструктуре EGI (адаптер glite);
- приложение, запускаемое на кластере под управлением системы TORQUE (адаптер cluster)

4.3 Дескриптор сервиса

Дескриптор сервиса оформляется в формате JSON и состоит из двух частей:

- "внешнее" описание сервиса, передаваемое клиентам (текстовая аннотация сервиса, описания входных и выходных параметров);
- "внутренняя" конфигурация сервиса, содержащая информацию о реализации сервиса (адаптер и его конфигурационные параметры).

Общий вид дескриптора сервиса:

```
{
    // Описание сервиса
    "name": "имя сервиса",
    "description": "текстовая аннотация",
    "inputs": {
        // описание входных параметров сервиса
    },
    "outputs": {
        // описание выходных параметров сервиса
    },

    // Конфигурация сервиса
    "config": {
        "input-files": {
            // соответствия между входными параметрами и файлами
            // в виде пар "параметр": "файл"
            // (перед запуском адаптера будет создан соотв. файл)
        }
        "output-files": {
            // соответствия между выходными параметрами и файлами
            // в виде пар "параметр": "файл"
            // (значение выходного параметра будет взято из соотв. файла)
            //
            // допускается маска (*) в конце имени файла (например, output.*)
            // в этом случае будет выбран первый файл с указанным префиксом
        }
        // Конфигурация адаптера
        "adapter": {
            "type": "имя адаптера",
            // набор атрибутов зависит от адаптера
            // (см. документацию к используемому адаптеру)
        }
    }
}
```

Для удобства редактирования дескриптора сервиса допускаются следующие отступления от спецификации JSON:

- Для повышения читабельности длинных текстовых значений можно делать переносы строк (переносы будут автоматически убраны при чтении дескриптора

контейнером)

4.4 Описание параметров

Для описания параметров используется спецификация JSON Schema. Формально атрибуты “inputs” и “outputs” в описании сервиса соответствуют значениям атрибутов “properties” в JSON-схемах, определяющих форматы запроса к сервису и результата вычислений.

Примеры описаний параметров можно найти в дескрипторах сервисов в EVEREST_HOME/services/examples/.

Внимание! Имена параметров не должны содержать точек. Во избежание проблем рекомендуется использовать в именах параметров только латинские буквы, цифры, подчеркивание и дефис.

4.5 Размещение сервиса в контейнере

Для размещения сервиса в контейнере необходимо выполнить следующие действия:

1. Создать директорию EVEREST_HOME/services/SERVICE_ID (допускаются произвольные вложенные директории внутри /services)
2. Разместить в данной директории файл service.json, содержащий дескриптор сервиса;
3. Разместить дополнительные файлы, связанные с сервисом (см. документацию к используемому адаптеру)
4. Перезапустить контейнер

После размещения сервиса можно проверить доступность сервиса по адресу

```
http(s)://HOST:PORT/services/SERVICE_ID
```

Адаптеры

5.1 Адаптер command

Command-адаптер адаптер предназначен для трансляции запроса к сервису, развернутому в контейнере Everest, в запуск определенной команды в отдельном процессе операционной системы.

Адаптер обеспечивает преобразование входных параметров запроса к сервису в аргументы командной строки и входные файлы команды, запуск команды на локальной машине, контроль над выполнением команды, а также обратное преобразование потоков вывода и выходных файлов команды в выходные параметры запроса к сервису по завершению выполнения команды.

Основное назначение адаптера - быстрое преобразование в сервисы существующих приложений с интерфейсом командной строки.

5.1.1 Установка и конфигурация

Адаптер входит в состав дистрибутива Everest. Установка дополнительного ПО не требуется.

Для загрузки command-адаптера при запуске Everest в секцию adapters конфигурационного файла EVEREST_HOME/conf/everest.conf необходимо добавить следующий элемент (включен в конфигурацию по-умолчанию):

```
{
    ...
    "adapters": [
        ...
        {
            "name": "command",
            "factory": "org.mathcloud.everest.adapter.command.CommandAdapterFactory"
        }
```

```
    ]
}
```

5.1.2 Конфигурация адаптера

Конфигурация command-адаптера имеет вид:

```
"adapter": {
    "type": "command",
    "command": "Шаблон запускаемой команды" // Обязательно
}
```

Атрибут command

Шаблон запускаемой команды с указанием мест подстановки значений входных параметров в виде $имя_параметра$.

Например:

```
"command": "myprogram $input1$ $input2$"
```

5.1.3 Примеры

Примеры реализованных command-сервисов можно найти в EVEREST_HOME/services/examples/:

- **echo**: демонстрируется работа с различными типами данных (параметров), вызывает команду echo, не работает под Windows
- **povray**: пример превращения в сервис реального приложения - POV-Ray
- **sleep**: тестовый сервис, запускающий команду sleep на заданное время

5.1.4 Детали реализации

Алгоритм обработки запроса к сервису с помощью command-адаптера состоит из следующих шагов:

- Формирование запускаемой команды, путем вставки значений входных параметров запроса в шаблон команды, указанный в конфигурации сервиса;
- Запуск команды в отдельном процессе операционной системы в рабочей директории задания, с перенаправлением стандартных потоков вывода в файлы stdout и stderr;
- Ожидание завершения процесса команды;
- Проверка кода завершения процесса (ненулевой код приводит к переводу задания в состояние FAILED и завершению обработки запроса)

В любой момент обработки запрос может быть отменен. В случае, если в данный момент запущен внешний процесс с командой, адаптер завершает выполнение процесса.

При возникновении ошибок в процессе обработки запроса адаптер переводит задание в состояние FAILED и завершает обработку запроса.

Внимание! В случае ошибок или отмены задания адаптер завершает только порожденный им процесс, но не его дочерние процессы. Если запускаемое адаптером приложение или скрипт порождает дочерние процессы, необходимо самостоятельно реализовать их завершение при завершении родительского процесса. В Linux для этого можно использовать команду trap, например следующим образом:

```
#!/bin/bash
trap 'jobs -p | xargs -r kill' 0
sleep $*
```

При завершении процесса, выполняющего данный скрипт, произойдет завершение всех порожденных им процессов (в данном примере - команды sleep).

5.2 Адаптер java

Java-адаптер предназначен для трансляции запроса к сервису, развернутому в контейнере Everest, в вызов определенного Java-класса в рамках используемой контейнером виртуальной машины Java.

Адаптер обеспечивает передачу входных параметров запроса к сервису при вызове Java-класса и прием значений выходных параметров в качестве результата вызова. Для этого используемый Java-класс должен реализовывать стандартный программный интерфейс.

Основное назначение адаптера - быстрое преобразование в сервисы существующих Java-приложений.

5.2.1 Установка и конфигурация

Адаптер входит в состав дистрибутива Everest. Установка дополнительного ПО не требуется.

Для загрузки Java-адаптера при запуске Everest в секцию adapters конфигурационного файла EVEREST_HOME/conf/everest.conf необходимо добавить следующий элемент (включен в конфигурацию по-умолчанию):

```
{
    ...
    "adapters": [
        ...
        {
            "name": "java",
```

```
            "factory": "org.mathcloud.everest.adapter.java.JavaAdapterFactory"
        }
    ]
}
```

5.2.2 Конфигурация адаптера

Конфигурация Java-адаптера имеет вид:

```
"adapter": {
    "type": "java",
    "class": "Полное имя Java-класса реализации сервиса", // Обязательно
    "params": { // Необязательно
        // произвольные пары ключ-значение,
        // передаваемые реализации сервиса при инициализации
    }
}
```

5.2.3 Реализация сервиса

Java-класс, реализующий обработку запросов к сервису, должен реализовывать интерфейс org.mathcloud.everest.adapter.java.JavaServiceI.

Метод init() предназначен для передачи сервису произвольных конфигурационных параметров в виде пар строк “ключ-значение”. Данные параметры указываются в атрибуте “params” конфигурации адаптера.

Метод run() предназначен для обработки запроса к сервису и имеет единственный аргумент типа Job.

Метод destroy() предназначен для уведомления реализации сервиса о свёртывании сервиса, например - при остановке контейнера.

При реализации Java-сервиса в IDE достаточно добавить в проект библиотеки

- EVEREST_HOME/lib/everest-core-x.x.x.jar
- EVEREST_HOME/lib/everest-java-x.x.x.jar
- EVEREST_HOME/lib/jettison-x.x.x.jar
- EVEREST_HOME/lib/slf4j-api-x.x.x.jar (при использовании логгирования)

Значения входных параметров в run() можно получить с помощью метода getInput(). Значения передаются в виде объекта типа org.codehaus.jettison.json.JSONObject. Это означает, что контейнер не делает никаких преобразований JSON в объекты Java, оставляя разработчику сервиса возможность работать с JSON-данными с помощью стандартных средств (библиотек Jettison и Jackson).

При работе с входными значениями разработчик может использовать следующие подходы:

- работать с JSONObject напрямую, извлекая из него значения параметров с помощью методов get..();
- использовать библиотеку Jackson для преобразования JSON-строки в определенные им Java-типы (POJO).

Аналогично устроена работа с выходными параметрами. Контейнер оперирует с ними уже в виде JSONObject и поддерживает только элементарные преобразования ограниченного круга типов Java (см. ниже) в JSON.

Значения выходных параметров можно передать с помощью методов

- Job.setOutput(JSONObject output) - передаются целиком все значения
- setOutput(String name, Object value) - передается значение одного параметра, допускается использование только следующих типов значения: Boolean, Double, Integer, JSONArray, JSONObject, Long, String, JSONObject.NULL.

При работе с выходными значениями разработчик может использовать следующие подходы

- создать JSONObject, устанавить значения параметров вручную с помощью методов JSONObject.put() и затем передать результаты с помощью первого метода;
- передавать выходные значения по-отдельности с помощью второго метода;
- использовать библиотеку Jackson для преобразования определенных им Java-типов (POJO) в JSON-строку и затем в JSONObject, передаваемый контейнеру с помощью первого метода.

В случае возникновения ошибок в процессе обработки запроса, реализация run() может выбрасывать исключения. Данные исключения будут перехвачены адаптером, в результате чего задание будет переведёно в состояние FAILED.

5.2.4 Размещение сервиса в контейнере

Реализацию сервиса необходимо упаковать в jar-файл и разместить его в директории сервиса EVEREST_HOME/services/SERVICE_ID вместе с дескриптором.

5.2.5 Примеры

Примеры реализованных Java-сервисов можно найти в EVEREST_HOME/services/examples/:

- **echo-java**: аналог command-сервиса echo, демонстрируется работа с различными типами данных (параметров) и использование библиотеки Jackson (исходный код)
- **fib**: демонстрируется работа с JSONObject напрямую, использование атрибута “params”, обработка прервывания (отмены задания) и логгирование (исходный код)

5.2.6 Детали реализации

Алгоритм обработки запроса к сервису с помощью Java-адаптера состоит из следующих шагов:

- Вызов указанного в конфигурации Java-класса (единственный экземпляр на контейнер) в отдельном потоке, с передачей входных параметров запроса к сервису;
- Ожидание завершения вызова и получений значений выходных параметров в качестве результата вызова;

В любой момент обработки запрос может быть отменен. В случае, если в данный момент выполняется вызов Java-класса, адаптер пытается прекратить его выполнение с помощью механизма прерывания потока.

При возникновении ошибок-исключений в процессе обработки запроса задание переводится в состояние FAILED.