Если вы видите что-то необычное, просто сообщите мне.

Run a Java Application as a Service on Linux

Введение

Любое Java приложение с точки зрения системы это просто объект JVM. В этом коротком руководстве мы увидим, как мы можем сделать наше приложение сервисом.

Мы будем истользовать удобства пакета system, systemd это сервис управления системой в современных дистрибутивах Linux.

Здесь вы найдёте две реализации: одна для простого случая, другая - расширенная.

Простой сервис

В мире systemd, для создания системного сервиса, нам нужно подготовить файл и зарегистрировать его определённым способом. Начнём с содержания файла:

```
[Unit]
Description=My Java driven simple service
After=syslog.target network.target

[Service]
SuccessExitStatus=143

User=appuser
Group=appgroup
```

```
Type=simple

Environment="JAVA_HOME=/path/to/jvmdir"
WorkingDirectory=/path/to/app/workdir
ExecStart=${JAVA_HOME}/bin/java -jar javaapp.jar
ExecStop=/bin/kill -15 $MAINPID

[Install]
WantedBy=multi-user.target
```

Мы узнали тип сервиса простым потому что система начинает JVM процесс напрямую без создания дочернего процесса.

ExecStop указывает команду завершения, и systemd достаточно умен чтобы выяснить PID начального процесса. он автоматически создаёт MAINPID переменные окружения.

После, мы указываем systemd посылать 15 (SIGTERM) системный сигнал, чтобы завершить процесс.

Java создатели спроектировали его таким образом, чтобы он возвращал не нулевой код в случае если он завершён системным сигналом. Так как сигнал не нулевой, то ответ будет `128 + числовое значение сигнала`.

У казывая SuccessExitStatus как 143, мы говорим systemd отловить это значение(128+15) как нормальное завершение

Форкаем сервис

Простой файл описания сервиса выше может быть довольно эффективным для простого приложения. Однако, во многих практических случаях, будут возможно включены дополнительные настройки.

Это может быть JVM параметры так же как любой другой параметр приложения, для примера, файл конфигурации или данных. Это модно свести к написанию обёртки для shell скрипта, где мы можем настроить все требуемые параметры прежде чем запустить JVM.

Предоставим, мы уже имеем обертку для скрипта, и теперь просто хотим включить это в сервис систем:

```
#!/bin/bash

JAVA_HOME=/path/to/jvmdir
WORKDIR=/path/to/app/workdir
JAVA_OPTIONS=" -Xms256m -Xmx512m -server "
APP_OPTIONS=" -c /path/to/app.config -d /path/to/datadir "

cd $WORKDIR
"${JAVA_HOME}/bin/java" $JAVA_OPTIONS -jar javaapp.jar $APP_OPTIONS
```

Так как мы используем shell скрипт чтобы запустить сервис, JVM будет запущена с помощью shell скрипта. Эта операция известна как `fork`, и поэтому мы указали тип как `forking`.

Перенесём опередления переменных в тело скрипта:

```
[Unit]
Description=My Java forking service
After=syslog.target network.target
[Service]
SuccessExitStatus=143
User=appuser
Group=appgroup

Type=forking

ExecStart=/path/to/wrapper
ExecStop=/bin/kill -15 $MAINPID

[Install]
WantedBy=multi-user.target
```

Регистрируем и запускаем сервис

Не важно какой тип сервиса выбран, для выполнения задачи, мы должны знать, как настроить и запустить сам systemd сервис.

First, we need to name the unit file after the service name we want to have. In our examples, that could be javasimple.service or javaforking.service.

Then, we put the unit file under one of the locations where systemd can find it. For an arbitrary service, /etc/systemd/system is a good choice.

The full path to our system units, in that case, will be:

```
/etc/systemd/system/javasimple.service
/etc/systemd/system/javaforking.service
```

Another possible path to place system units is /usr/lib/systemd/system. This is typically the location used by the system installation packages.

However, we should consider it more appropriate when we develop our own .rpm or .deb installation packages containing system services.

In either case, we'll control the service using the systemctl utility and pass either the start, stop, or status command.

Before that, however, we should notify systemd that it has to rebuild its internal service database. Doing this will make it aware of the new system unit we introduced. We can do this by passing the daemon-reload command to systemctl.

Now, we're ready to run all the commands we mentioned:

```
sudo systemctl daemon-reload

sudo systemctl start javasimple.service
```

```
sudo systemctl status javasimple.service

● javasimple.service - My Java driven simple service
Loaded: loaded (/etc/systemd/system/javasimple.service; disabled; vendor preset: disabled)
Active: active (running) since Sun 2021-01-17 20:10:19 CET; 8s ago
Main PID: 8124 (java)
CGroup: /system.slice/javasimple.service
└─8124 /path/to/jvmdir/bin/java -jar javaapp.jar
```

We'll need to run the daemon-reload command each time we modify the unit file.

Next, we notice the system reports our service running but disabled. Disabled services will not start automatically when the system boots.

Of course, we can configure it to start up automatically along with the system. This is where we use another systemctl command — enable:

```
sudo systemctl enable javasimple.service
Created symlink from /etc/systemd/system/multi-user.target.wants/javasimple.service to
/etc/systemd/system/javasimple.service
```

Now, we can see that it's enabled:

```
sudo systemctl status javasimple.service
● javasimple.service - My Java driven simple service
Loaded: loaded (/etc/systemd/system/javasimple.service; enabled; vendor preset: disabled)
Active: active (running) since Sun 2021-01-17 20:10:19 CET; 14min ago
Main PID: 8124 (java)
....
```

5. Conclusion

In this article, we looked at two possible ways of turning Java applications into system service by means of systemd.

Java is still one of the most popular programming languages. A lot of Java applications are designed to run non-interactively for a variety of tasks, such as processing data, providing an API, monitoring events, and so on. Thus, they all are good candidates to become system services.

Revision #3

Created 18 April 2021 10:56:42 by gasick

Updated 16 April 2023 19:38:12 by gasick