Systemd Watchdog for Any Service

Making basic systemd service is easy. Let's assume the simplest application (not necessarily even designed to be a service) and look into making it work with systemd.

Our example application will be a script in `/opt/test/application` with the following content:

```
#!/bin/bash

while(true); do
  date | tee /var/tmp/test.log
  sleep 1
done
```

Essentially it's just never ending output of a current date.

To make it a service, we simply create `/etc/systemd/system/test.service` with description of our application:

```
[Unit]
Description=Test service
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
ExecStart=/opt/test/application
Restart=always
RestartSec=1

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

That's all needed before we can start the service:

```
sudo systemctl start test

sudo systemctl status test
 ● test.service - Test service
    Loaded: loaded (/etc/systemd/system/test.service; disabled; vendor preset: enabled)
    Active: active (running)
  Main PID: 5212 (service)
     Tasks: 2 (limit: 4657)
    CGroup: /system.slice/test.service
```

```
           ├─5212 /bin/bash /opt/test/application
           └─5321 sleep 1
```

Systemd will start application and even perform restart if application fails. But what if we want it a bit smarter? What if we want a watchdog that'll restart application not only when it's process fails but also when some other health check goes bad?

While sytemd does support such setup, application generally should be aware of it and call watchdog function every now and then. Fortunately, even if our application doesn't do that, we can use watchdog facilities via `systemd-notify` tool.

First we need to change three things in our service definition. One is changing type to `notify`, then changing executable to the wrapper script, and lastly defining the watchdog time.

In this example, if application doesn't respond in 5 seconds, it will be considered failed. The new service definition in `/etc/systemd/system/test.service` can look something like this:

```
[Unit]
Description=Test service
After=network.target
StartLimitIntervalSec=0

[Service]
Type=^^notify^^
ExecStart=^^/opt/test/test.sh^^
Restart=always
RestartSec=1
TimeoutSec=5
WatchdogSec=^^5^^

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

Those watching carefully will note we don't actually solve anything with this and that we just move all responsibility to `/opt/test/test.sh` wrapper.

It's in that script we first communicate to sytemd when application is ready and later, in a loop, check for not only application PID but also for any other condition (e.g. certain curl response), calling **`systemd-notify`** if application proves to be healthy:

```
#!/bin/bash

trap 'kill $(jobs -p)' EXIT

/opt/test/service &
PID=$!

/bin/systemd-notify --ready

while(true); do
    FAIL=0

    kill -0 $PID
    if [[ $? -ne 0 ]]; then FAIL=1; fi

#    curl http://localhost/test/
#    if [[ $? -ne 0 ]]; then FAIL=1; fi

    if [[ $FAIL -eq 0 ]]; then /bin/systemd-notify WATCHDOG=1; fi

    sleep 1
done
```

Starting service now gives slightly different output:

```
sudo systemctl stop test

sudo systemctl start test

sudo systemctl status test
 ● test.service - Test service
    Loaded: loaded (/etc/systemd/system/test.service; disabled; vendor preset: enabled)
    Active: active (running)
  Main PID: 6406 (test.sh)
     Tasks: 4 (limit: 4657)
    CGroup: /system.slice/test.service
            ├─6406 /bin/bash /opt/test/test.sh
            ├─6407 /bin/bash /opt/test/application
            ├─6557 sleep 1
            └─6560 sleep 1
```

If we kill application manually (e.g. **`sudo kill 6407`**), systemd will pronounce service dead and start it again. It will do the same if any other check fails.

While this approach is not ideal, it does allow for easy application watchdog retrofitting.