

# Aktor: spacermake (Primary-Secondary mit Nutzungsprotokoll)

Dieser Aktor ist eine Zuarbeit vom [Makerspace Bocholt](#). Der Aktor agiert zusammen mit [Aktor: Machine Logger \(CSVlog\)](#) und dient dazu, sekundär eingeschaltene Geräte in einem Log zu erfassen, damit Verbräuche zu tracken und verwendet dabei das DIY-Hardwarekonzept [Primary-Secondary Schaltung](#). Der Aktor schickt außerdem Anzeigeinformationen an ggf. installierte [FabReader](#) oder [FabCounter](#)-Anzeigen. spacermake greift deshalb unter anderem auf die bestehende [FabFire Adapter Konfiguration](#) (`config.toml`) zurück.

## spacermake installieren

Wir klonen das Projekt:

```
mkdir -p /opt/fabinfra/adapters/  
git clone https://github.com/LastExceed/spacermake.git  
cd spacermake/
```

Vor dem Kompilieren müssen wir noch ein paar Pfade für Konfigurations- und Logdateien anpassen. Das Script ist aktuell nicht optimal konfigurierbar und muss deshalb vor dem Nutzen geeignet angepasst und erst dann kompiliert werden!

```
vim /src/main.rs
```

```
# config Pfade anpassen  
static ref SLAVES_BY_MASTER: HashMap<String, HashSet<String>> = parse_toml_file("master-  
slave_relations.toml");  
static ref SLAVE_PROPERTIES: HashMap<String, [bool; 3]> = parse_toml_file("slave_properties.toml");  
static ref MACHINE_IDS: HashMap<String, String> =  
parse_toml_file::<toml::Table>("/opt/fabinfra/adapters/fabfire_adapter/config/config.toml")  
  
...  
  
# und den MQTT Server anpassen (wir fügen auch noch eine Zeile für Benutzer und Passwort ein)  
let mut mqttoptions = MqttOptions::new("spacermake", "localhost", 1883);
```

```
mqttoptions.set_credentials("fabinfra101", "fablocal");
```

```
vim /src/utls/logs.rs
```

```
.open("machinelog.csv")?
```

```
# und weiter unten:
```

```
.open("machinelog_debug.csv")?
```

Wir installieren Rust, falls noch nicht vorhanden ist:

```
# Wir installieren nun das aktuelle Rust per rustup (als normaler Nutzer). Rustup erlaubt das flexible Installieren beliebiger Rust-Versionen
```

```
curl https://sh.rustup.rs -sSf | sh
```

```
# cargo in .bashrc einfügen und Umgebung neu laden
```

```
echo 'source "$HOME/.cargo/env"' >> ~/.bashrc
```

```
source ~/.bashrc
```

```
# wir prüfen, ob wir die aktuelle Rust Version haben
```

```
rustup show
```

```
# oder installieren sie ...
```

```
rustup install stable
```

```
rustup default stable
```

Dann erzeugen wir die Binary:

```
cd /opt/fabinfra/adapters/spacermake/
```

```
cargo build --release
```

## spacermake konfigurieren und testen

```
cd /opt/fabinfra/adapters/spacermake/
```

## Log Files

```
# leere Log Files anlegen. Sonst startet spacermake nicht
```

```
touch machinelog.csv
```

```
touch machinelog_debug.csv
```

## Primary-Secondary (Master-Slave) konfigurieren

Die folgenden beiden \*.toml Dateien müssen konfiguriert werden. Eine Beispielkonfiguration:

```
vim master-slave_relations.toml
```

```
master1 = ["slave1", "slave2"]  
master2 = ["slave3", "slave1"]
```

```
vim slave_properties.toml
```

```
slave1 = [false, false, true]  
slave2 = [true, true, true]
```

Die in `slave_properties.toml` angegebenen Boolean-Werte definieren folgendes:

- `runsContinuously` - Slave läuft immer (`true`) oder soll von Buchungs bis Rückgabe des Masters laufen (`false`)
- `needsTrailingTime` - Slave soll nach Abschalten des Masters 30 Sekunden nachlaufen (`true`) oder Secondary geht sofort aus (`false`)
- `isTasmota` - gibt an, ob es ein Tasmota (`true`) oder Shelly (`false`) ist

## Berechtigungen anpassen

Wir übergeben die Dateien alle dem Nutzer `bffh`:

```
cd /opt/fabinfra/adapters/  
chown -R bffh:bffh spacermake/
```

## Manuell prüfen

Wir prüfen manuell, ob die Binary startet:

```
/opt/fabinfra/adapters/spacermake/target/release/spacermake
```

## spacermake als systemd Service

```
sudo vim /etc/systemd/system/spacermake.service
```

#### [Unit]

Description=FabAccess Primary-Secondary Actor with usage log protocol

Require=network-online.target

After=network-online.target

#### [Service]

Type=simple

User=bffh

Group=bffh

ExecStart=/opt/fabinfra/adapters/spacermake/target/release/spacermake

Restart=always

WorkingDirectory=/opt/fabinfra/adapters/spacermake

#### [Install]

WantedBy=multi-user.target

Wir aktualisierne den Daemon, aktivieren und starten den Dienst dann:

```
sudo systemctl daemon-reload  
sudo systemctl enable spacermake.service --now
```

Die Logs finden wir dann mit:

```
sudo journalctl -f -u spacermake.service
```

---

Version #14

Erstellt: 24 Februar 2025 01:28:07 von Mario Voigt (Stadtfabrikanten e.V.)

Zuletzt aktualisiert: 3 März 2025 23:09:35 von Mario Voigt (Stadtfabrikanten e.V.)