LEITFADEN

VDA5050-Prüfungen in der Build-Pipeline

November 2026 · Für Entwickler von Fahrzeugsoftware und Leitsteuerungen

Eine Build-Pipeline kann LogIQ Fleets auf zwei Arten nutzen:

- **Einen Mitschnitt prüfen.** Die Pipeline schickt den MQTT-Verkehr eines Tests, etwa aus einer Simulation, und bekommt Urteil und Funde zurück. Ein Fahrzeug ist nicht nötig, und von der Datei wird nichts gespeichert.
- **Einen Zertifizierungslauf fahren.** Die Pipeline startet einen geplanten Zertifizierungslauf mit einem echten Fahrzeug am Broker, wartet auf den Versuch und seinen Bericht und bekommt beides zurück.

Beides endet gleich: ein Urteil auf der Konsole, eine JUnit-XML-Datei für das CI-System und ein Rückgabewert, der den Build bestehen oder scheitern lässt. Ein Werkzeug erledigt beides.

Was Sie brauchen

	Mitschnitt prüfen	Zertifizierungslauf fahren
Anmeldung	API-Schlüssel `lxk_…`	persönliches Token `lxp_…`
Ausgestellt von	der Administration der Installation, auf Anfrage	Ihnen selbst, unter Mein Konto → Tokens für Pipelines
Variable	`LOGIQ_API_KEY`	`LOGIQ_TOKEN`
Außerdem	ein Mitschnitt des Verkehrs	ein geplanter Lauf, für Pipelines freigegeben, und eine Verbindung zum Broker des Fahrzeugs

Beide brauchen die Adresse der Installation in `LOGIQ_URL`: die Adresse, unter der Sie sich anmelden, etwa `https://logiq-fleets.com`.

Das Werkzeug

Jede Installation liefert das Werkzeug unter `/static/downloads/logiq_check.py` aus. Es braucht nur die Python-Standardbibliothek und läuft damit in jeder Pipeline mit Python 3.9 oder neuer.

```
curl -fsSO https://<installation>/static/downloads/logiq_check.py
python3 logiq_check.py --print-version
```

Laden Sie das Werkzeug von der Installation, gegen die Sie prüfen: Beide werden gemeinsam ausgeliefert. Sie können die Datei auch in Ihr Repository legen, müssen sie dann aber von Hand aktuell halten.

Einen Mitschnitt prüfen

```
LOGIQ_URL=https://<installation> LOGIQ_API_KEY=lxk_… \
  python3 logiq_check.py run.log --lif layout.lif.json --junit vda5050.xml
```

Der Mitschnitt. Das Werkzeug nimmt NDJSON, ein JSON-Array, einen Capture-Export von LogIQ Fleets oder die Ausgabe von `mosquitto_sub -v`, mit der Endung `.ndjson`, `.json`, `.log` oder `.txt`. Am einfachsten zeichnen Sie einen Test auf, indem Sie die Topics des Fahrzeugs abonnieren, solange der Test läuft:

```
mosquitto_sub -h localhost -t 'uagv/#' -v > run.log &
RECORDER=$!
./run-simulation.sh            # Ihr Test: Aufträge, Sofortaktionen, ...
kill $RECORDER
```

Der Mitschnitt sollte beide Richtungen enthalten: die Aufträge und Sofortaktionen der Leitsteuerung und die State-, Connection- und Factsheet-Nachrichten des Fahrzeugs. Eine Prüfung bewertet nur, was der Mitschnitt enthält. Situationen, die nicht vorkamen, stehen als *nicht gezeigt* in der Antwort und zählen nie als bestanden.

Die Optionen.

Option	Bedeutung
`--lif DATEI`	ein LIF-Layout; wird selbst geprüft und dient als Karte, gegen die die Aufträge geprüft werden
`--factsheet DATEI`	das Factsheet des Fahrzeugs; wird selbst geprüft und dient als Maßstab der Fähigkeitsprüfungen
`--version`	die VDA5050-Version, nach der geprüft wird (1.1.0, 2.0.0, 2.1.0, 3.0); Standard: die, die der Mitschnitt angibt
`--junit DATEI`	das Ergebnis als JUnit-XML schreiben
`--fail-on`	was den Build scheitern lässt, durch Komma getrennt: `fail` (Standard), `warning`, `inconclusive`; `never` für nichts
`--strict`	einen Mitschnitt mit unlesbaren Zeilen ablehnen
`--lang`	Sprache der Funde: en (Standard), de, zh
`--timeout`	Sekunden, die auf die Antwort gewartet wird (Standard 300)
`--quiet`	nur die Zeile mit dem Urteil ausgeben

Ein Layout oder ein Factsheet lässt sich auch ohne Mitschnitt prüfen: `python3 logiq_check.py --lif layout.lif.json --junit lif.xml`.

Grenzen. Standardmäßig nimmt eine Prüfung bis zu 20 MB und 20.000 Nachrichten, ein Schlüssel bis zu 30 Prüfungen pro Minute, eine zur Zeit, und 200.000 Nachrichten pro Tag. Die Administration der Installation kann die Grenzen ändern. Eine abgelehnte Prüfung endet mit Rückgabewert 2 und sagt, warum.

Einen Zertifizierungslauf fahren

Ein Zertifizierungslauf bewegt ein echtes Fahrzeug. Die Pipeline darf deshalb nur einen Lauf starten, den jemand für sie freigegeben hat, und nur über eine Verbindung, die es schon gibt.

1. **Den Lauf planen**, wie gewohnt in der Testbench. Ein Lauf mit Schritten, die eine Person ausführen muss, etwa einem Gang zum Fahrzeug, lässt sich nicht freigeben. Die Kurzabnahme braucht niemanden am Fahrzeug und passt gut in eine Pipeline.
2. **Den Lauf freigeben.** Auf der Seite des Laufs, unter *Aus einer Pipeline*, bestätigt jemand, dass die Anlage im Testbetrieb bleibt und das Fahrzeug bewegt werden darf, ohne dass jemand am Bildschirm sitzt, und drückt *Für Pipelines freigeben*. Die Seite nennt, wer wann freigegeben hat. *Freigabe zurücknehmen* beendet die Freigabe.
3. **Ein Token anlegen**, unter Mein Konto → Tokens für Pipelines. Es wird einmal angezeigt. Es handelt als Ihr Konto und kann nur Läufe starten, die für Pipelines freigegeben sind.
4. **Die Verbindung wählen.** Die Pipeline nennt sie mit ihrer ID oder ihrem Namen. Nehmen Sie eine Verbindung, die die Installation selbst erreicht. Eine Verbindung über ein Browserfenster funktioniert nur, solange dieses Fenster offen ist.

```
LOGIQ_URL=https://<installation> LOGIQ_TOKEN=lxp_… \
  python3 logiq_check.py --certify <Lauf-ID> --broker <Verbindung> --junit certification.xml
```

Das Werkzeug startet einen Versuch, fragt alle fünf Sekunden nach dem Stand und wartet auf den Bericht, der dem Versuch folgt, standardmäßig bis zu einer Stunde (`--wait` in Sekunden). Dann gibt es aus, was gezeigt wurde, was nicht und warum, und das Urteil des Berichts mit seinem Link.

Von einem Lauf kann immer nur ein Versuch zur Zeit gefahren werden. Ein zweiter Start wird abgelehnt, bis der erste beendet ist.

Rückgabewerte

Wert	Mitschnitt prüfen	Zertifizierungslauf fahren
0	in Ordnung: konform oder nicht schlechter, als `--fail-on` zulässt	jedes Szenario gezeigt, Bericht in Ordnung
1	nicht konform	ein Szenario, das die Version verlangt, wurde nicht gezeigt, oder der Bericht ist nicht in Ordnung
2	die Prüfung wurde nicht durchgeführt: Netz, abgelehnt, zu groß	nichts entscheidbar: nicht gestartet, Versuch gescheitert, ein Szenario konnte nicht abgefragt werden, kein Bericht innerhalb von `--wait`
3	falscher Aufruf	falscher Aufruf

Ein Szenario, das nicht gezeigt wurde, für das seine VDA5050-Version aber keine Antwort festlegt, ist eine Warnung. Es lässt den Build nur mit `--fail-on warning` scheitern.

Was die JUnit-Datei enthält

Eine Prüfung: ein Testfall je Prüfung, eine Test-Suite je Gruppe, dazu Suiten für das Layout, GB/T 47864 und Ihre eigenen Regelwerke, wo es sie gibt.

Ergebnis der Prüfung	In JUnit
FAIL	`failure`, mit den ersten fünf Funden
PASS	bestanden
WARNING	bestanden; die Funde stehen in `system-out`, weil JUnit keine Warnung kennt
keine Daten, abgeschaltet	`skipped`

Ein Zertifizierungslauf: zuerst eine Suite *Scenarios* mit einem Testfall je Szenario, das der Lauf zeigen sollte. Ein nicht gezeigtes Szenario ist ein `failure` mit dem Grund seines Segments, eines, das nicht abgefragt werden konnte, ist `skipped`. Danach folgen die Suiten des Berichts wie bei einer Prüfung.

Beispiele

Das Werkzeug endet mit 1, wenn das Ergebnis nicht in Ordnung ist. Der Schritt, der die JUnit-Datei veröffentlicht, muss deshalb trotzdem laufen. Jedes Beispiel unten sorgt dafür.

GitHub Actions

```
name: VDA5050
on: [push]

jobs:
  vda5050:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Record a simulated shift
        run: ./ci/record-shift.sh run.log
      - name: Check the recording
        env:
          LOGIQ_URL: ${{ vars.LOGIQ_URL }}
          LOGIQ_API_KEY: ${{ secrets.LOGIQ_API_KEY }}
        run: |
          curl -fsSO "$LOGIQ_URL/static/downloads/logiq_check.py"
          python3 logiq_check.py run.log --lif layout.lif.json --junit vda5050.xml
      - name: Keep the result
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: vda5050
          path: vda5050.xml
```

Auf GitHub Actions setzt das Werkzeug selbst Anmerkungen: Jeder Fund einer Critical- oder Major-Prüfung wird ein Fehler am Build, Minor und Warning eine Warnung, und eine Tabelle geht in die Job-Zusammenfassung. Bei einem Zertifizierungslauf wird jedes nicht gezeigte Szenario, das seine Version verlangt, ein Fehler, die übrigen eine Warnung. Außerhalb von GitHub schaltet `--github` das ein.

GitLab CI

```
vda5050:
  image: python:3.12-slim
  script:
    - apt-get update && apt-get install -y --no-install-recommends curl mosquitto-clients
    - ./ci/record-shift.sh run.log
    - curl -fsSO "$LOGIQ_URL/static/downloads/logiq_check.py"
    - python3 logiq_check.py run.log --lif layout.lif.json --junit vda5050.xml
  artifacts:
    when: always
    reports:
      junit: vda5050.xml
```

`LOGIQ_URL` und `LOGIQ_API_KEY` gehören unter Settings → CI/CD → Variables, der Schlüssel maskiert. GitLab zeigt dann jede Prüfung im Testbericht des Merge Requests.

Jenkins

```
pipeline {
  agent any
  environment {
    LOGIQ_URL = 'https://<installation>'
    LOGIQ_TOKEN = credentials('logiq-pipeline-token')
  }
  stages {
    stage('Certification run') {
      steps {
        sh 'curl -fsSO "$LOGIQ_URL/static/downloads/logiq_check.py"'
        sh 'python3 logiq_check.py --certify <run id> --broker <connection> --junit
certification.xml'
      }
    }
  }
  post {
    always { junit 'certification.xml' }
  }
}
```

Azure Pipelines

```
steps:
  - script: |
      curl -fsSO "$(LOGIQ_URL)/static/downloads/logiq_check.py"
      python3 logiq_check.py run.log --junit vda5050.xml
    env:
      LOGIQ_URL: $(LOGIQ_URL)
      LOGIQ_API_KEY: $(LOGIQ_API_KEY)
  - task: PublishTestResults@2
    condition: always()
    inputs:
      testResultsFormat: JUnit
      testResultsFiles: vda5050.xml
```

Was wann läuft

Pipeline	Passt für
jeder Push oder Merge Request	die Prüfung des Mitschnitts eines simulierten Tests: Minuten, kein Fahrzeug
nachts oder vor einem Release	einen Zertifizierungslauf mit einem Fahrzeug am Testbroker
ein neues Layout oder Factsheet	die Prüfung der Datei allein mit `--lif` oder `--factsheet`

Schlüssel und Tokens

- Legen Sie Schlüssel und Tokens im Secret-Speicher des CI-Systems ab, nie im Repository. `--key` und `--token` funktionieren auch, sind auf der Kommandozeile aber in der Prozessliste sichtbar.
- Einen Schlüssel oder ein Token, das nicht mehr gebraucht wird, widerrufen Sie: Schlüssel über die Administration der Installation, Tokens selbst unter Mein Konto. Ein widerrufenes wird sofort abgelehnt.
- Ein Token handelt als das Konto, das es angelegt hat. Legen Sie eines je Pipeline an, damit sich eines widerrufen lässt, ohne die anderen anzuhalten.

Wenn etwas abgelehnt wird

Meldung	Ursache
`unauthorized` / *This token is unknown or has been revoked*	falscher oder widerrufener Schlüssel oder Token, oder die falsche Installation
`throttled`, `busy`, `budget`	eine Grenze des Schlüssels; die Meldung sagt, welche
`too_large`	mehr, als die Installation in einer Prüfung annimmt
`unreadable`	die Datei enthält keine VDA5050-Nachrichten, die die Prüfung lesen kann
`not_released`	der Lauf ist nicht für Pipelines freigegeben
`needs_somebody`	der Lauf hat Schritte, die eine Person ausführen muss
`busy` (Zertifizierung)	ein Versuch dieses Laufs wird schon gefahren
No connection … / *… connections are called …*	der Name der Verbindung ist unbekannt oder nicht eindeutig; geben Sie ihre ID an
`not_started`	der Lauf konnte nicht starten, etwa weil der Broker nicht erreichbar war; die Meldung sagt, warum

Weiterlesen

- Partner-Prüf-API: die API hinter der Prüfung, ihre Antworten und die JSON-Felder.
- Zertifizierungsläufe: einen Lauf planen, seine Segmente und das Urteil eines Versuchs.

Fragen zur CI-Anbindung erreichen uns unter **support@logiq-fleets.com**.