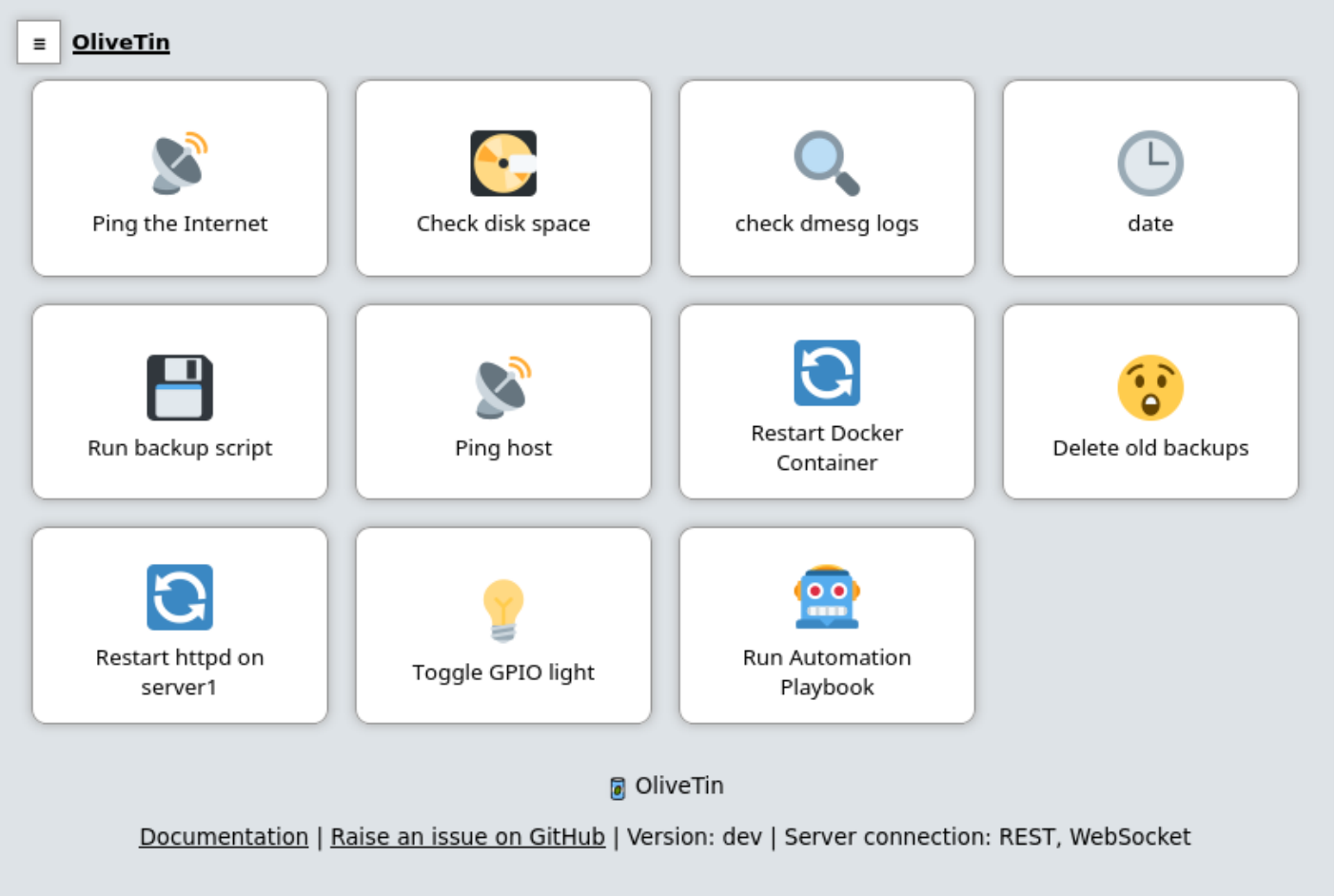
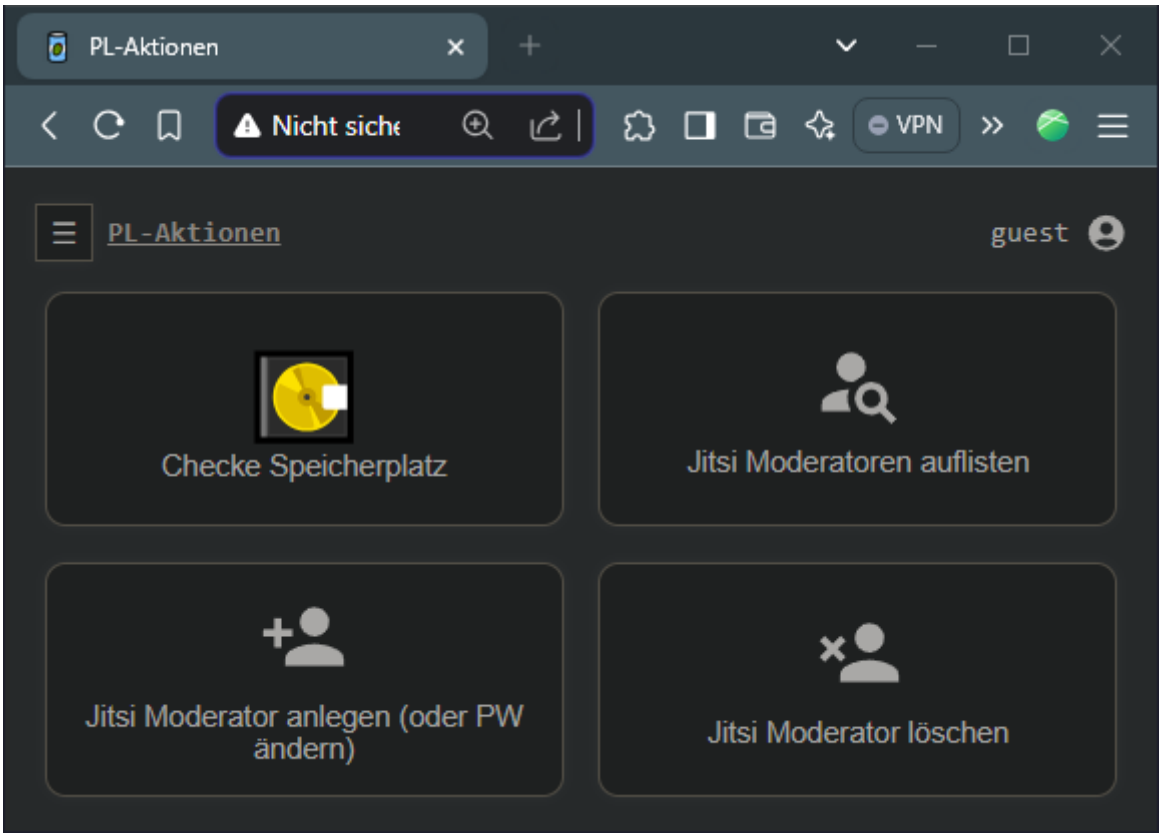
OliveTin (Web-UI für Shell-Cmds)

OliveTin





https://github.com/OliveTin/OliveTin

Eikes docker-compose.yml (mit Möglichkeit für docker exec cmds)

```
version: "3.8"
services:
  olivetin:
    container_name: olivetin
    image: jamesread/olivetin:2024.11.18
    user: root # wichtig für 'docker exec... Befehle in anderen Containern'
    volumes:
      - cfg:/config
      - /var/run/docker.sock:/var/run/docker.sock # wichtig für 'docker exec... Befehle in
anderen Containern'
    ports:
      - "1337:1337"
    restart: unless-stopped

volumes:
  cfg:
```

Eikes config.yaml für Jitsi mit Authentik

```
# There is a built-in micro proxy that will host the webui and REST API all on
# one port (this is called the "Single HTTP Frontend") and means you just need
# one open port in the container/firewalls/etc.
#
# Listen on all addresses available, port 1337
listenAddressSingleHTTPFrontend: 0.0.0.0:1337

# Choose from INFO (default), WARN and DEBUG
logLevel: "DEBUG"

# Checking for updates https://docs.olivetin.app/update-checks.html
checkForUpdates: false

pageTitle: Shell-Aktionen
showFooter: false

# Actions are commands that are executed by OliveTin, and normally show up as
# buttons on the WebUI.
#
# Docs: https://docs.olivetin.app/create-your-first-action.html
actions:
  # This uses `popupOnStart: execution-dialog-stdout-only` to simply show just
  # the command output.
  - title: Checke Speicherplatz
    icon: disk
    shell: df -h /media
    popupOnStart: execution-dialog-stdout-only
    # ACL MUSS NICHT MEHR BEI JEDER AKTION EINGESTELLT WERDEN, WEIL
    # addToEveryAction: true
    # BEI DEN ACL EINSTELLUNGEN UNTEN
    # acls:
    #   - Authentik-Admins
    #   - AdminsNachUsername



  - title: Jitsi Moderatoren auflisten
    icon: <iconify-icon icon="mdi:user-search"></iconify-icon>
    shell: docker exec jitsi-prosody-1 sh -c "find /config/data/meet%2ejitsi/accounts -type f
-exec basename {} .dat \; | sed -e 's/%2e/./g' -e 's/%2d/-/g' -e 's/%5f/_/g'"
```

```
    popupOnStart: execution-dialog-stdout-only


  - title: Jitsi Moderator anlegen (oder PW ändern)
    #shell: echo Test für execution-dialog-stdout-only: User: {{ beuntzername }} PW: {{
passwort }}
    shell: docker exec jitsi-prosody-1 sh -c "prosodyctl --config /config/prosody.cfg.lua
register {{ beuntzername }} meet.jitsi {{ passwort }}"
    icon: <iconify-icon icon="mdi:user-add"></iconify-icon>
    timeout: 100
    popupOnStart: execution-dialog-stdout-only
    arguments:
      - name: beuntzername
        title: Benutzername
        type: ascii_identifier
        default: m.oderator
        description: Verwende nur "-", "_" und ".", keine anderen Sonderzeichen, Umlaute oder
Leerzeichen
      - name: passwort
        title: Passwort
        type: ascii_identifier
        default: Passwort123
        description: PW abspeichern! Verwende nur "-", "_" und ".", keine anderen
Sonderzeichen, Umlaute oder Leerzeichen


  - title: Jitsi Moderator löschen
    shell: docker exec jitsi-prosody-1 sh -c "prosodyctl --config /config/prosody.cfg.lua
unregister {{ beuntzername }} meet.jitsi"
    icon: <iconify-icon icon="mdi:user-remove"></iconify-icon>
    timeout: 100
    popupOnStart: execution-dialog-stdout-only
    arguments:
      - name: beuntzername
        title: Benutzername
        type: ascii_identifier
        default: m.oderator
        description: Der Name des Moderators zb m.oderator

authRequireGuestsToLogin: true # Optional - depends if you want to "disable" guests.
```

```
authOAuth2RedirectURL: "https://olivetin.MEINEDOMAIN.de/oauth/callback"
authOAuth2Providers:
  authentik:
    name: authentik
    title: Authentik
    clientID: "GWF9.......YaHj"
    clientSecret: "3yOkhOMCA..............84xOpK2Y1gHzihyyMxkv5"
    authURL: "https://auth.MEINEDOMAIN.de/application/o/authorize/"
    tokenURL: "https://auth.MEINEDOMAIN.de/application/o/token/"
    whoamiURL: "https://auth.MEINEDOMAIN.de/application/o/userinfo/"
    usernameField: "preferred_username"
    icon: <iconify-icon icon="simple-icons:authentik"></iconify-icon>


defaultPermissions:
 view: false
 exec: false
 logs: true

# GROUPS KLAPPT LEIDER NICHT, SIEHE
# https://github.com/OliveTin/OliveTin/issues/477
# DAHER MATCHUSERNAMES MIT DEN EINZELEN AUTHENTIK-USERN (AdminsNachUsername)
accessControlLists:
  - name: Authentik-Admins
    matchUsergroups:
      - Authentik-Admins
    permissions:
      view: true
      exec: true
  - name: AdminsNachUsername
    addToEveryAction: true
    matchUserNames:
      - demo-admin
      - vorname.nachname
      - m.mustermann
    permissions:
      view: true
      exec: true
```

Authentik OIDC

https://docs.olivetin.app/oauth2-authentik.html

Original config.yaml mit Beispielen

```
# There is a built-in micro proxy that will host the webui and REST API all on
# one port (this is called the "Single HTTP Frontend") and means you just need
# one open port in the container/firewalls/etc.
#
# Listen on all addresses available, port 1337
listenAddressSingleHTTPFrontend: 0.0.0.0:1337

# Choose from INFO (default), WARN and DEBUG
logLevel: "INFO"

# Checking for updates https://docs.olivetin.app/update-checks.html
checkForUpdates: false

# Actions are commands that are executed by OliveTin, and normally show up as
# buttons on the WebUI.
#
# Docs: https://docs.olivetin.app/create-your-first-action.html
actions:
  # This is the most simple action, it just runs the command and flashes the
  # button to indicate status.
  #
  # If you are running OliveTin in a container remember to pass through the
  # docker socket! https://docs.olivetin.app/action-container-control.html
  - title: Ping the Internet
    shell: ping -c 3 1.1.1.1
    icon: ping
    popupOnStart: execution-dialog-stdout-only

  # This uses `popupOnStart: execution-dialog-stdout-only` to simply show just
  # the command output.
  - title: Check disk space
    icon: disk
    shell: df -h /media
    popupOnStart: execution-dialog-stdout-only
```

```
  # This uses `popupOnStart: execution-dialog` to show a dialog with more
  # information about the command that was run.
  - title: check dmesg logs
    shell: dmesg | tail
    icon: logs
    popupOnStart: execution-dialog

  # This uses `popupOnStart: execution-button` to display a mini button that
  # links to the logs.
  - title: date
    shell: date
    timeout: 6
    icon: clock
    popupOnStart: execution-button

  # You are not limited to operating system commands, and of course you can run
  # your own scripts. Here `maxConcurrent` stops the script running multiple
  # times in parallel. There is also a timeout that will kill the command if it
  # runs for too long.
  - title: Run backup script
    shell: /opt/backupScript.sh
    shellAfterCompleted: "apprise -t 'Notification: Backup script completed' -b 'The backup
script completed with code {{ exitCode}}. The log is: \n {{ output }} '"
    maxConcurrent: 1
    timeout: 10
    icon: backup
    popupOnStart: execution-dialog

  # When you want to prompt users for input, that is when you should use
  # `arguments` - this presents a popup dialog and asks for argument values.
  #
  # Docs: https://docs.olivetin.app/action-ping.html
  - title: Ping host
    shell: ping {{ host }} -c {{ count }}
    icon: ping
    timeout: 100
    popupOnStart: execution-dialog-stdout-only
    arguments:
      - name: host
        title: Host
```

```
      type: ascii_identifier
      default: example.com
      description: The host that you want to ping

    - name: count
      title: Count
      type: int
      default: 3
      description: How many times to do you want to ping?

# OliveTin can control containers - docker is just a command line app.
#
# However, if you are running in a container you will need to do some setup,
# see the docs below.
#
# Docs: https://docs.olivetin.app/action-container-control.html
- title: Restart Docker Container
  icon: restart
  shell: docker restart {{ container }}
  arguments:
    - name: container
      title: Container name
      choices:
        - value: plex
        - value: traefik
        - value: grafana

# There is a special `confirmation` argument to help against accidental clicks
# on "dangerous" actions.
#
# Docs: https://docs.olivetin.app/confirmation.html
- title: Delete old backups
  icon: ashtonished
  shell: rm -rf /opt/oldBackups/
  arguments:
    - type: confirmation
      title: Are you sure?!

# This is an action that runs a script included with OliveTin, that will
# download themes. You will still need to set theme "themeName" in your config.
```

```
#
# Docs: https://docs.olivetin.app/themes.html
- title: Get OliveTin Theme
  shell: olivetin-get-theme {{ themeGitRepo }} {{ themeFolderName }}
  icon: theme
  arguments:
    - name: themeGitRepo
      title: Theme's Git Repository
      description: Find new themes at https://olivetin.app/themes
      type: url

    - name: themeFolderName
      title: Theme's Folder Name
      type: ascii_identifier

# Sometimes you want to run actions on other servers - don't overcomplicate
# it, just use SSH! OliveTin includes a helper to make this easier, which is
# entirely optional. You can also setup SSH manually.
#
# Docs: https://docs.olivetin.app/action-ssh-easy.html
# Docs: https://docs.olivetin.app/action-ssh.html
- title: "Setup easy SSH"
  icon: ssh
  shell: olivetin-setup-easy-ssh
  popupOnStart: execution-dialog

# Here's how to use SSH with the "easy" config, to restart a service on
# another server.
#
# Docs: https://docs.olivetin.app/action-ssh-easy.html
# Docs: https://docs.olivetin.app/action-service.html
- title: Restart httpd on server1
  id: restart_httpd
  icon: restart
  timeout: 1
  shell: ssh -F /config/ssh/easy.cg root@server1 'service httpd restart'

# Lots of people use OliveTin to build web interfaces for their electronics
# projects. It's best to install OliveTin as a native package (eg, .deb), and
# then you can use either a python script or the `gpio` command.
```

```
- title: Toggle GPIO light
  shell: gpioset gpiochip1 9=1
  icon: light

# There are several built-in shortcuts for the `icon` option, but you
# can also just specify any HTML, this includes any unicode character,
# or a <img = "..." /> link to a custom icon.
#
# Docs: https://docs.olivetin.app/icons.html
#
# Lots of people use OliveTin to easily execute ansible-playbooks. You
# probably want a much longer timeout as well (so that ansible completes).
#
# Docs: https://docs.olivetin.app/ansible-playbook.html
- title: "Run Automation Playbook"
  icon: '&#129302;'
  shell: ansible-playbook -i /etc/hosts /root/myRepo/myPlaybook.yaml
  timeout: 120

# The following actions are "dummy" actions, used in a Dashboard. As long as
# you have these referenced in a dashboard, they will not show up in the
# `actions` view.
- title: Ping hypervisor1
  shell: echo "hypervisor1 online"

- title: Ping hypervisor2
  shell: echo "hypervisor2 online"

- title: "{{ server.name }} Wake on Lan"
  shell: echo "Sending Wake on LAN to {{ server.hostname }}"
  entity: server

- title: "{{ server.name }} Power Off"
  shell: "echo 'Power Off Server: {{ server.hostname }}'"
  entity: server

- title: Ping All Servers
  shell: "echo 'Ping all servers'"
  icon: ping
```

```
  - title: Start {{ container.Names }}
    icon: box
    shell: docker start {{ container.Names }}
    entity: container
    trigger: Update container entity file

  - title: Stop {{ container.Names }}
    icon: box
    shell: docker stop {{ container.Names }}
    entity: container
    trigger: Update container entity file

  # Lastly, you can hide actions from the web UI, this is useful for creating
  # background helpers that execute only on startup or a cron, for updating
  # entity files.

  # - title: Update container entity file
  #   shell: 'docker ps -a --format json > /etc/OliveTin/entities/containers.json'
  #   hidden: true
  #   execOnStartup: true
  #   execOnCron: '*/1 * * * *'

# An entity is something that exists - a "thing", like a VM, or a Container
# is an entity. OliveTin allows you to then dynamically generate actions based
# around these entities.
#
# This is really useful if you want to generate wake on lan or poweroff actions
# for `server` entities, for example.
#
# A very popular use case that entities were designed for was for `container`
# entities - in a similar way you could generate `start`, `stop`, and `restart`
# container actions.
#
# Entities are just loaded fome files on disk, OliveTin will also watch these
# files for updates while OliveTin is running, and update entities.
#
# Entities can have properties defined in those files, and those can be used
# in your configuration as variables. For example; `container.status`,
# or `vm.hostname`.
#
```

```
# Docs: http://docs.olivetin.app/entities.html
entities:
  # YAML files are the default expected format, so you can use .yml or .yaml,
  # or even .txt, as long as the file contains valid a valid yaml LIST, then it
  # will load properly.
  #
  # Docs: https://docs.olivetin.app/entities.html
  - file: entities/servers.yaml
    name: server

  - file: entities/containers.json
    name: container

# Dashboards are a way of taking actions from the default "actions" view, and
# organizing them into groups - either into folders, or fieldsets.
#
# The only way to properly use entities, are to use them with a `fieldset` on
# a dashboard.
dashboards:
  # Top level items are dashboards.
  - title: My Servers
    contents:
      - title: All Servers
        type: fieldset
        contents:
          # The contents of a dashboard will try to look for an action with a
          # matching title IF the `contents: ` property is empty.
          - title: Ping All Servers

          # If you create an item with some "contents:", OliveTin will show that as
          # directory.
          - title: Hypervisors
            contents:
              - title: Ping hypervisor1
              - title: Ping hypervisor2

      # If you specify `type: fieldset` and some `contents`, it will show your
      # actions grouped together without a folder.
      - type: fieldset
        entity: server
```

```
        title: 'Server: {{ server.hostname }}'
        contents:
          # By default OliveTin will look for an action with a matching title
          # and put it on the dashboard.
          #
          # Fieldsets  also support `type: display`, which can display arbitary
          # text. This is useful for displaying things like a container's state.
          - type: display
            title: |
              Hostname: <strong>{{ server.name }}</strong>
              IP Address: <strong>{{ server.ip }}</strong>

          # These are the actions (defined above) that we want on the dashboard.
          - title: '{{ server.name }} Wake on Lan'
          - title: '{{ server.name }} Power Off'

  # This is the second dashboard.
  - title: My Containers
    contents:
      - title: 'Container {{ container.Names }} ({{ container.Image }})'
        entity: container
        type: fieldset
        contents:
          - type: display
            title: |
              {{ container.RunningFor }} <br /><br /><strong>{{ container.State }}</strong>

          - title: 'Start {{ container.Names }}'
          - title: 'Stop {{ container.Names }}'
```

config.yaml Beispiel für Jitsi Meet Moderator anlegen

```
actions:
  - title: Jitsi Moderator anlegen
      #shell: echo {{ beuntzername }} meet.jitsi {{ passwort }}
      shell: docker exec jitsi-prosody-1 sh -c "prosodyctl --config /config/prosody.cfg.lua
register {{ beuntzername }} meet.jitsi {{ passwort }}"
      icon: ping
      timeout: 100
      popupOnStart: execution-dialog-stdout-only
```

```
    arguments:
      - name: beuntzername
        title: Benutzername
        type: ascii_identifier
        default: m.oderator
        description: Der Name des Moderators zb m.oderator

      - name: passwort
        title: Passwort
        type: ascii_identifier
        default: Passwort123
        description: Das Passwort (keine Leerzeichen, PW abspeichern!)
```

Revision #13
Created 2024-11-20 08:19:20 UTC
Updated 2024-11-20 21:37:29 UTC