Runp

runp contributors

Version 0.7.0

Overview

Like `docker-compose` but not just for containers.

Today, every non-trivial software project is compound of a number of parts (applications, libraries, services) running together in a lot of environments: virtual machines, Docker containers, physical boxes, cloud...

Runp allows you to specify what and how you want to run and helps you to start the system as a whole, running all needed processes in parallel (like Docker compose, Foreman et simili).

This speeds up the setup of a working environment compared to the usual approaches like custom scripts, documentation (written or spoken), or, more commonly, leaving it to the developer to figure out.

Runp uses a `Runpfile`, a sort of a sofisticated `Procfile` that describes the project as a system composed of `units` each of which can run a different type of process such as:

- processes on the physical box
- container processes
- SSH tunnel processes

Moreover the Runpfile handles:

- working directory per process
- environment variables per process
- user defined vars per system, defined in the Runpfile or passed at runtime as command line arguments
- processes dependency: a process can wait for a given condition to start, e.g. a file appearing or another process being ready
- not only *long running* processes but *one shot commands* too
- Windows OS

An example of a `Runpfile` :

```
name: Example
description: |
  Sample Runpfile to show runp functionalities
units:
  web:
    description: Web app
    # this process is running on host machine
    host:
      command: node app.js
      workdir: backend
      env:
        # inherit PATH from host system to find needed tools (e.g. node)
```

```
      PATH: $PATH
    await:
      # wait for the DB being available
      resource: tcp4://localhost:5432/
      timeout: 0h0m10s
mail:
  description: Test mail server
  # this process is running in a container
  container:
    image: docker.io/mailhog/mailhog
    ports:
      - "8025:8025"
      - "1025:1025"
db:
  description: Corporate DB
  # This process is reachable through SSH port forwarding
  ssh_tunnel:
    user: user
    auth:
      identity_file: ~/.ssh/id_rsa
    local:
      port: 5432
    jump:
      host: dev.host
      port: 22
    target:
      host: corporate.db
      port: 5432
```

Usage

Run runp

Some commands:

```
runp --help                          # generic help
runp help up                         # describes the command "up"
runp up                              # run the runpfile in the current directory
runp -d up -f /path/to/runpfile.yaml # run in debug mode processes in the given
Runpfile
runp encrypt --key test secret       # encrypt "secret" using the key "test" and print
                                     # out the value to use in a Runpfile
runp ls -f /path/to/runpfile.yaml    # list units in Runpfile
```

Settings

Runp looks for a settings file in `~/.runp/settings.yaml`.

If the file does not exist or is empty/invalid, Runp starts using these defaults:

```
container_runner: docker
```

Preconditions

Any unit is ran if all its preconditions are satisfied.

Available preconditions:

- OS: unit is ran if OS is the specified one.
- Runp version: unit is ran if Runp version is in range.
- Hosts: unit is ran if `/etc/hosts` contains the given mapping.

Runpfile composition

Runpfile can include other Runpfiles:

```
name: Test Runpfile
description: This is Runpfile
include:
  - Runpfile-env.yml
  - Runpfile-vars.yml
```

App waiting for DB

A backend app running on host waiting for a DB running in a container to be available:

```
units:
  be:
    description: Backend app
    host:
      command: mvn clean compile quarkus:dev
      workdir: backend
      env:
        # inherit PATH from host system to find mvn and java
        PATH: $PATH
      await:
        resource: tcp4://localhost:5432/
        timeout: 0h0m10s
  db:
    description: Database
    container:
      image: docker.io/postgres:alpine
      ports:
        - "5432:5432"
      env:
        POSTGRES_PASSWORD: pass
        POSTGRES_USER: user
        POSTGRES_DB: dbname
```

Containers

You can set the container engine using the settings file (key: `container_runner`).

Example:

```
container_runner: /path/to/podman
```

WARNING	Only Docker and Podman (as they use the same command line flags) are supported.

Containers can talk to each other thorough a Docker network (`runp-network`).

The container name (the host name exposed to other containers) is set to `runp-${UNIT NAME}` or to the field `name`.

This Runpfile starts Wordpress and MySql:

```
name: Wordpress Runpfile
description: Runpfile to run Wordpress and MySql
units:
  db:
    container:
      name: db
      image: docker.io/mysql:5.7
```

```
    ports:
      - "3306:3306"
    env:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    container:
      image: docker.io/wordpress:latest
      ports:
        - "8000:80"
      env:
        WORDPRESS_DB_HOST: db:3306
        WORDPRESS_DB_USER: wordpress
        WORDPRESS_DB_PASSWORD: wordpress
        WORDPRESS_DB_NAME: wordpress
      await:
        resource: tcp4://localhost:3306/
        timeout: 0h0m20s
```

Use containers volumes

Run containers and volumes (example is from the book Docker in action - Manning):

```
name: Containers Runpfile
description: This is Runpfile showing Docker volumes
units:
  fowler:
    description: The Fowler collection
    container:
      image: docker.io/alpine:3.12
      skip_rm: true
      mounts:
        - "type=volume,dst=/library/PoEAA"
        - "type=bind,src=/tmp,dst=/library/DSL"
      command: |
        echo "Fowler collection created"
  knuth:
    description: The Knuth collection
    container:
      image: docker.io/alpine:3.12
      skip_rm: true
      mounts:
        - "type=volume,dst=/library/TAoCP.vol1"
        - "type=volume,dst=/library/TAoCP.vol2"
        - "type=volume,dst=/library/TAoCP.vol3"
      command: |
        echo "Knuth collection created"
  reader:
```

```
    description: The avid reader
    container:
      image: alpine:3.12
      volumes_from:
        - fowler
        - knuth
      command: |
        ls -l /library/
      await:
        timeout: 0h0m3s
```

On Windows

Windows is supported:

```
name: Test Runpfile
description: This is Runpfile
units:
  await:
    description: read environment variables
    host:
      command: set
      env:
        # in env block variables have always the unix notation
        MYHOME: ${HOME}
  echo:
    description: echo the value of %OS% env var
    host:
      # when used in command, env vars have the specific OS notation
      command: echo %OS%
  infiniteloop:
    description: infinite loop
    host:
      # this script is in examples/ directory
      executable: infinite.cmd
      workdir: examples
```

Some programs, especially on Windows, implicitly use OS specific environment variables.

If you run into strange problems try adding these to the env block:

```
# Windows env vars
SystemRoot: ${SystemRoot}
ALLUSERSPROFILE: ${ALLUSERSPROFILE}
APPDATA: ${APPDATA}
CommonProgramFiles: ${CommonProgramFiles}
CommonProgramW6432: ${CommonProgramW6432}
ComSpec: ${ComSpec}
DriverData: ${DriverData}
```

```
HOMEDRIVE: ${HOMEDRIVE}
HOMEPATH: ${HOMEPATH}
LOCALAPPDATA: ${LOCALAPPDATA}
OS: ${OS}
PATHEXT: ${PATHEXT}
ProgramData: ${ProgramData}
ProgramFiles: ${ProgramFiles}
ProgramW6432: ${ProgramW6432}
PSModulePath: ${PSModulePath}
PUBLIC: ${PUBLIC}
SESSIONNAME: ${SESSIONNAME}
SystemDrive: ${SystemDrive}
TEMP: ${TEMP}
TMP: ${TMP}
USERNAME: ${USERNAME}
USERPROFILE: ${USERPROFILE}
windir: ${windir}
```

Run a different command on different operative systems

Inclusions are compared to `runtime.GOOS`:

```
units:
  win:
    description: Windows unit
    preconditions:
      os:
        # this unit is ran when os is windows
        inclusions:
          - windows
    host:
      command: dir {{vars runp_root}}
  unix:
    description: Nix unit
    preconditions:
      os:
        # this unit will be ran when os is linux or darwin
        inclusions:
          - linux
          - darwin
    host:
      command: ls -al {{vars runp_root}}
```

SSH tunnel to reach a remote LDAP

A backend app running on host using LDAP on remote server available using SSH tunneling.

SSH tunnel can use three auth methods:

- `identity_file`: the path to the private key, ie `~/.ssh/id_rsa`

- `secret`: the SSH server password in plain text
- `encrypted_secret`: the SSH server password encrypted and in base 64 (you can create it using `runp encrypt`)

```
units:
  be:
    description: Backend app
    host:
      command: mybackendapp
      workdir: backend
  ldap:
    description: LDAP
    ssh_tunnel:
      user: runp
      auth:
        #identity_file: ~/tmp/runpssh/ssh/runp
        #secret: "plain text secret"
        encrypted_secret: "NsM1hcAy/L2TfACgfzbhYyb9j5a2ySYcARFDKkv7HTk="
      local:
        # localhost is the default
        port: 389
      jump:
        host: sshserver
        port: 22
      target:
        host: ldapserver
        port: 389
```

Use secrets

SSH tunnel process allows user to use secrets to specify the password.

To create the encrypted secret:

```
runp encrypt -k thekey SECRET
```

The above command will encrypt the string `SECRET` using the password `thekey`.

To run a Runpfile containing an `encrypted_secret` you have to pass the key to the `up` command (the key must coincide with the one used to encrypt).

You can pass the key on command line using the options `--key` or `--key-env`

Using the `-k`/`--key` argument the key is in plain text on the command line:

```
runp up -k thekey
```

Use the `--key-env` argument Runp looks up for that environment variable and use its value as key:

```
runp up --key-env RUNP_SECRET
```

Use environment variables

A one-shot command using custom environment variables:

```
  env3:
    description: echo command
    host:
      command: echo ${MYHOME}
      workdir: ..
      env:
        MYHOME: ${HOME}
```

User defined variables

Use runtime vars:

```
vars:
  foo: FOO_DEFAULT_VALUE
units:
  vars-test-unit:
    description: echo a user defined var
    host:
      command: echo __{{vars foo}}__
```

The var `foo` will have value `FOO_DEFAULT_VALUE` or can be set from command line:

```
$ bin/runp --debug up -f examples/Runpfile-vars.yml --var foo=bar
```

Implicit variables

Runp adds to the context some variables:

- `runp_workdir`: user current working directory as absolute path
- `runp_root`: directory parent of the Runpfile as absolute path
- `runp_file_separator`: OS file separator (`/` on unix, `\` on windows)

Usage:

```
units:
  vars:
    description: echo implicit vars from Runp
```

```
    host:
      command: "echo runp_workdir={{vars runp_workdir}} runp_root={{vars runp_root}}"
```

Disabling color output

To have plain, non-colored text output set the environment variable `NO_COLOR`:

```
NO_COLOR=1 ./bin/runp -d up -f examples/Runpfile-many-units.yml
```

or use the option `--no-color`:

```
./bin/runp -d --no-color up -f examples/Runpfile-many-units.yml
```

Runpfile Runp version

A unit can require a constraint on the Runp version.

This unit requires runp version greater the 0.5.0:

```
units:
  test1:
    description: test unit
    preconditions:
      runp:
        operator: GreaterThan
        version: 0.5.0
    host:
      command: env
      workdir: ${HOME}
```

The available operators are:

- `LessThan`
- `LessThanOrEqual`
- `Equal`
- `GreaterThanOrEqual`
- `GreaterThan`

License