tmt

ENJOY CREATING, EXECUTING AND ENABLING TESTS

Miroslav Vadkerti • Petr Šplíchal • Pavel Valena

Nest with Fedora 2020

AGENDA

What to expect from this talk?

- Short introduction
- Real life examples
- More examples
- A bit of theory
- Even more examples
- CI infrastructure status
- How to get involved
- Questions (ask any time)
- Workshop tomorrow

introduction

WHO

Who are we?

- Petr Šplíchal
  - Operating System CI
  - psss / psplicha@redhat.com
- Miroslav Vadkerti
  - Testing Farm Team
  - thrix / mvadkert@redhat.com
- Pavel Valena
  - Ruby Maintenance Team
  - pvalena / pvalena@redhat.com

WHY

At the beginning there were user stories...

As a developer or tester I want:

- an easy way how to contribute tests
- easily run tests in my preferred environment
- easily reproduce issue revealed by the testing tool
- have more flexible test execution metadata stored at a single place
- unified, concise and human-friendly configuration for testing and gating

HOW

So what's the plan?

- Store all test execution metadata at one place
  - plain text, human readable, versioned under git, no external tcms
- Use an efficient format to store configuration, prevent duplication
  - fmf … flexible metadata format, yaml plus hierarchy, inheritance, elasticity
- Well define/design the test configuration syntax
  - specification … several metadata levels to cover well individual use cases
- Implement tool to make is easy to create, execute, debug, enable, maintain tests
  - tmt … test management tool, comfortable cli for everyday work
- Consistently use the same testing configuration across products
  - GitHub, RHEL, Fedora, CentOS…

the first steps

INSTALL

Get tmt on your laptop

```
# basic features, executing tests on localhost
sudo dnf install -y tmt

# additional dependencies for executing tests in containers
sudo dnf install -y tmt-provision-container

# running tests in a virtual machine using testcloud
sudo dnf install -y tmt-provision-virtual

# all available subpackages including all dependencies
sudo dnf install -y tmt-all
```

Enjoy creating, executing and enabling tests using tmt

simple

EXPLORE

Let’s look around!

```
tmt
```

INIT

Initialize the metadata tree

```
# create a minimal template
tmt init --template mini

# usually there is a short option as well
tmt init -t mini

# empty init
tmt init
```

SIMPLE

Simple use case should be super simple to write

```
summary: Basic smoke test
execute:
    script: tmt --help
```

MINI

The minimal config

```
execute:
    script: tmt --help
```

RUN

Run tests

```
tmt run
```

Enjoy creating, executing and enabling tests using tmt

RESULTS

What went wrong?

```
tmt run --last report -fvvv
```

--help is your friend

tmt run --help

How can I run tests?

```
Commands:

discover   Gather information about test cases to be executed.
provision  Provision an environment for testing or use localhost.
prepare    Prepare the environment for testing.
execute    Run tests using the specified framework.
report     Provide test results overview and send reports.
finish  Perform the finishing tasks, clean up guests.

login      Provide user with an interactive shell on the guest.
plans      Select plans which should be executed.
tests      Select tests which should be executed.
```

tmt run provision --help

Which provision options are available?

```
Supported methods (virtual by default):

connect ............... Connect to a provisioned guest using ssh
local ................. Use local host for test execution
minute ................ Provision guest using 1minutetip backend
container ............. Create a new container using podman
virtual.testcloud .... Local virtual machine using testcloud

Use 'tmt run provision --help --how <method>' to learn more
about given  provision method and all its supported options.
```

tmt run provision --how virtual --help

How can I customize the virtual method?

```
-h, --how METHOD   Use specified method for provisioning.
-i, --image IMAGE  Select image to use. Short name or url.
-m, --memory MEM   Set available memory, 2048 MB by default.
-D, --disk DISK    Specify disk size in GB, 10 GB by default.
-u, --user USER    Username to use for all guest operations.

-v, --verbose      Show more details. Use multiple times to raise
-d, --debug        Provide debugging information.
-q, --quiet        Be quiet. Exit code is just enough for me.
-f, --force        Overwrite existing files and step data.
-n, --dry          Run in dry mode. No changes, please.
```

tmt run

discover

DISCOVER

Just let me see which tests would be run

```
# only selected step will be run
tmt run discover

# discover tests only for given plan
tmt run discover plan --name /plans/basic

# show more details (list tests)
tmt run discover -v plan --name /plans/basic
tmt run discover --verbose plan --name /plans/basic
```

provision

CONTAINER

Run tests in container using podman

```
# choose custom provision method but run --all steps
tmt run --all provision --how container

# use custom image instead of the latest fedora default
tmt run --all provision --how container --image fedora:32
```

Enjoy creating, executing and enabling tests using tmt

VIRTUAL

Run tests in a virtual machine

```
# tests are run safely in a vm by default
# (the plan can define a different provision method)
tmt run

# this is the same as above
# (command line options override configuration in plan)
tmt run --all provision --how virtual

# use custom image instead of the latest fedora default
tmt run --all provision --how virtual --image fedora-32
```

Enjoy creating, executing and enabling tests using tmt

LOCAL

Execute tests directly on the localhost

```
# this is fast but destructive tests can break your box
tmt run --all provision --how local
```

CONNECT

Connect to a provisioned guest using ssh

```
# private key authentication (recommended)
tmt run --all provision --how=connect --guest=name-or-ip \
    --key=private-key-path

# user and password
tmt run --all provision --how=connect --guest=name-or-ip \
    --user=login --password=secret
```

prepare

INSTALL

Install additional packages on the guest

```
# packages from the default repositories
tmt run --all prepare --how install --package httpd

# latest package from a copr repository
tmt run -a prepare -h install --copr psss/tmt --package tmt

# freshly built local rpm / all rpms from directory
tmt run -a prepare -h install -p tmt-0.20-1.fc32.noarch.rpm
tmt run -a prepare -h install --directory tmp/RPMS/noarch

# check --help for detailed examples
tmt run prepare -h install --help
```

ANSIBLE

Apply ansible playbook on the guest

```
# apply one or more playbooks
tmt run -a prepare -h ansible -p ansible/packages.yml

# check --help for detailed examples
tmt run prepare -h install --help
```

SHELL

Run arbitrary shell command to setup the guest

```
# install apache and start the service
tmt run -a prepare --how shell \
    --script 'dnf install -y httpd && systemctl start httpd'
```

report

REPORT

Report test results

```
# by default a brief result overview is shown
tmt run plan --name plans/helps

# show summary for all plans from the last run
tmt run --last report

# show results of individual tests
tmt run --last report -fv

# list output files, show full output to terminal
tmt run --last report -fvv
tmt run --last report -fvvv
```

a bit of theory

fmf

FMF

Flexible Metadata Format

- Allows efficiently storing metadata in plain text files
- All test execution data directly in git, close to the test/source code
- Uses YAML to store data in a concise human and machine readable way
- Adds a couple of nice features to minimize data duplication and maintenance
  - Hierarchy
  - Inheritance
  - Elasticity
- https://fmf.readthedocs.io/

SIMPLE

The most common use cases super simple to read & write

```
summary: Check basic download options
contact: Petr Šplíchal <psplicha@redhat.com>
tag: [Tier2, TierSecurity]
test: ./runtest.sh
duration: 3m
```

HIERARCHY

Hierarchy defined by directory structure and explicit nesting

```
/download:
    summary: Check basic download options
    contact: Petr Šplíchal <psplicha@redhat.com>
    tags: [Tier2, TierSecurity]
    test: ./runtest.sh
    duration: 3m
/recursion:
    summary: Check recursive download options
    contact: Petr Šplíchal <psplicha@redhat.com>
    tags: [Tier2, TierSecurity]
    test: ./runtest.sh
    duration: 20m
```

INHERITANCE

Minimize duplication/maintenance by inheriting from parent

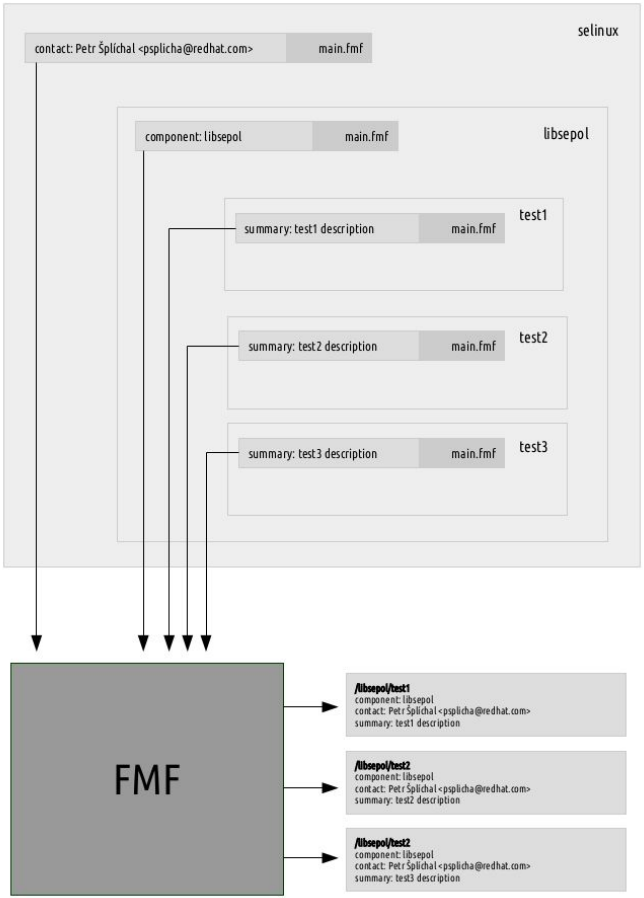
```
contact: Petr Šplíchal <psplicha@redhat.com>
tags: [Tier2, TierSecurity]
test: ./runtest.sh

/download:
    summary: Check basic download options
    duration: 3m
/recursion:
    summary: Check recursive download options
    duration: 20m
```

ELASTICITY

Allows to use a single file or scatter metadata across the hierarchy as desired by the project.

Nicely supports an "organic" growth of a project: Usually it is not possible to plan/predict in advance which parts of the system will grow more / will need more detailed configuration or related data to be stored.

Elasticity allows to move data areas which grow more into separate files on the filesystem, keeping in this way reasonable file size for maintaining the data while at the same time representing the whole metadata tree structure/content combined from all available files unchanged.



specification

METADATA LEVELS

There are several levels defined in the metadata specification

- **Level 0** — Core attributes such as summary for short overview, description for detailed texts or the order which are common and can be used across all metadata levels.
- **Level 1** — Metadata closely related to individual Tests such as the test script, directory path or maximum duration which are stored directly with the test code.
- **Level 2** — This level represents Plans made up of individual Steps describing how to provision the environment for testing and how to prepare it or which frameworks should be used to execute tests relevant for given artifact.
- **Level 3** — User Stories can be used to define expected features of the application and to easily track which functionality has been already implemented, tested and documented.

EXAMPLE TEST

```
summary: Basic smoke test for virtualenv
description: |
    Check basic functionality of Python virtual environments (venv
    or virtualenv based). The test supports different python
    versions & implementations including pypy and jython.
path: smoke
test: ./venv.sh
tier: 1
tag: [venv]
duration: 10m
require:
    - python3
    - python34
    - python35
    - python36
    ...
```

Enjoy creating, executing and enabling tests using tmt

EXAMPLE PLAN

```
summary: Basic httpd smoke test
provision:
    how: virtual
    memory: 4096
prepare:
  - name: packages
    how: install
    package: httpd curl
  - name: service
    how: shell
    script: systemctl start httpd
execute:
    how: shell
    script:
     - echo foo > /var/www/html/index.html
     - curl http://localhost/ | grep foo
```

Enjoy creating, executing and enabling tests using tmt

tmt plan

EXPLORE

Check which plans are available

```
# give an overview
tmt plans

# list available plans
tmt plans ls

# show plan details
tmt plans show

# let's see some tmt examples
vim -p plans/*
```

Enjoy creating, executing and enabling tests using tmt

CREATE

Easily create plans based on templates

```
# minimal config (execute a script)
tmt plan create plans/mini

# base plan (discover, execute)
tmt plan create plans/base -t base

# full plan (discover, prepare, execute)
tmt plan create plans/full -t full

# fill custom data directly from command line
tmt plan create --discover <yaml> --prepare <yaml>
```

Enjoy creating, executing and enabling tests using tmt

tmt test

EXPLORE

Check which tests are available

```
# give an overview
tmt tests

# list available tests
tmt tests ls
tmt tests ls --filter tier:0

# show test details
tmt tests show

# check test metadata against the spec
tmt test lint
```

Enjoy creating, executing and enabling tests using tmt

CREATE

Easily create tests based on templates

```
# prompt for template
tmt test create tests/smoke

# use a simple shell test template
tmt test create tests/smoke -t shell

# populate with a basic beakerlib skeleton
tmt test create tests/smoke -t beakerlib
```

Enjoy creating, executing and enabling tests using tmt

IMPORT & EXPORT

Convert test metadata from other formats

```
# convert beaker Makefile and PURPOSE
tmt test import

# fetch metadata from nitrate test case management system
tmt test import --nitrate

# sync back metadata to nitrate (fluent transition)
tmt test export --nitrate .

# try it yourself
cd examples/convert
```

tmt story

COVERAGE

Check implementation, test and documentation coverage

```
# explore and create same as with tests and plans
tmt stories
tmt stories ls
tmt stories show
tmt story create

# coverage overview
tmt story coverage
tmt story coverage cli/test
tmt story coverage --documented
tmt story coverage --implemented
tmt story coverage --tested
```

packit integration

PACKIT

Provides tooling and automation to integrate upstream open source projects into Fedora operating system

Packit service is a Github application which is able to automatically build Fedora COPR builds from PR's code.

https://github.com/marketplace/packit-as-a-service

Packit documentation

https://packit.dev/testing-farm/

The workflow

1. Packit service builds the PR in COPR for all configured chroots in .packit.yaml
2. Packit calls Testing Farm Service to conduct testing
2. VM image is chosen according to COPR chroot tested.
3. All builds from COPR are installed.
4. (optional) FMF tests are executed
5. Results from Testing Farm are displayed as additional PR checks

https://dashboard.packit.dev/jobs

next steps

Testing Farm

Testing system as a Service, shared across RHEL CI, Fedora CI and Packit with one public API.

Aiming to run:

- generic package level tests in TMT format
- functional tests in TMT format against a container
- functional tests in TMT format against a VM
- functional tests in Standard Test Interface (STI) format

fedora ci integration

The workflow

1. Fedora CI picks up a new update from bodhi checks dist-git for presence of STI or TMT tests
2. Fedora CI calls Testing Farm to conduct testing
3. Testing Farm installs all packages from the build
4. Tests are executed
5. Results from Testing Farm are displayed in the "Automated checks" tab next to the bodhi build

https://bodhi.fedoraproject.org/updates/FEDORA-2020-9044199e68

early access coming by the end of this month

feedback & contributions

Links

- tmt docs
  - https://tmt.readthedocs.io/
- fmf docs
  - https://fmf.readthedocs.io/
- fedora quick start guide
  - https://docs.fedoraproject.org/en-US/ci/tmt/
- packit integration docs
  - https://packit.dev/testing-farm/

Feedback / help

- Issue tracker: https://github.com/psss/tmt/issues

- IRC: #tmt on Freenode

thank you