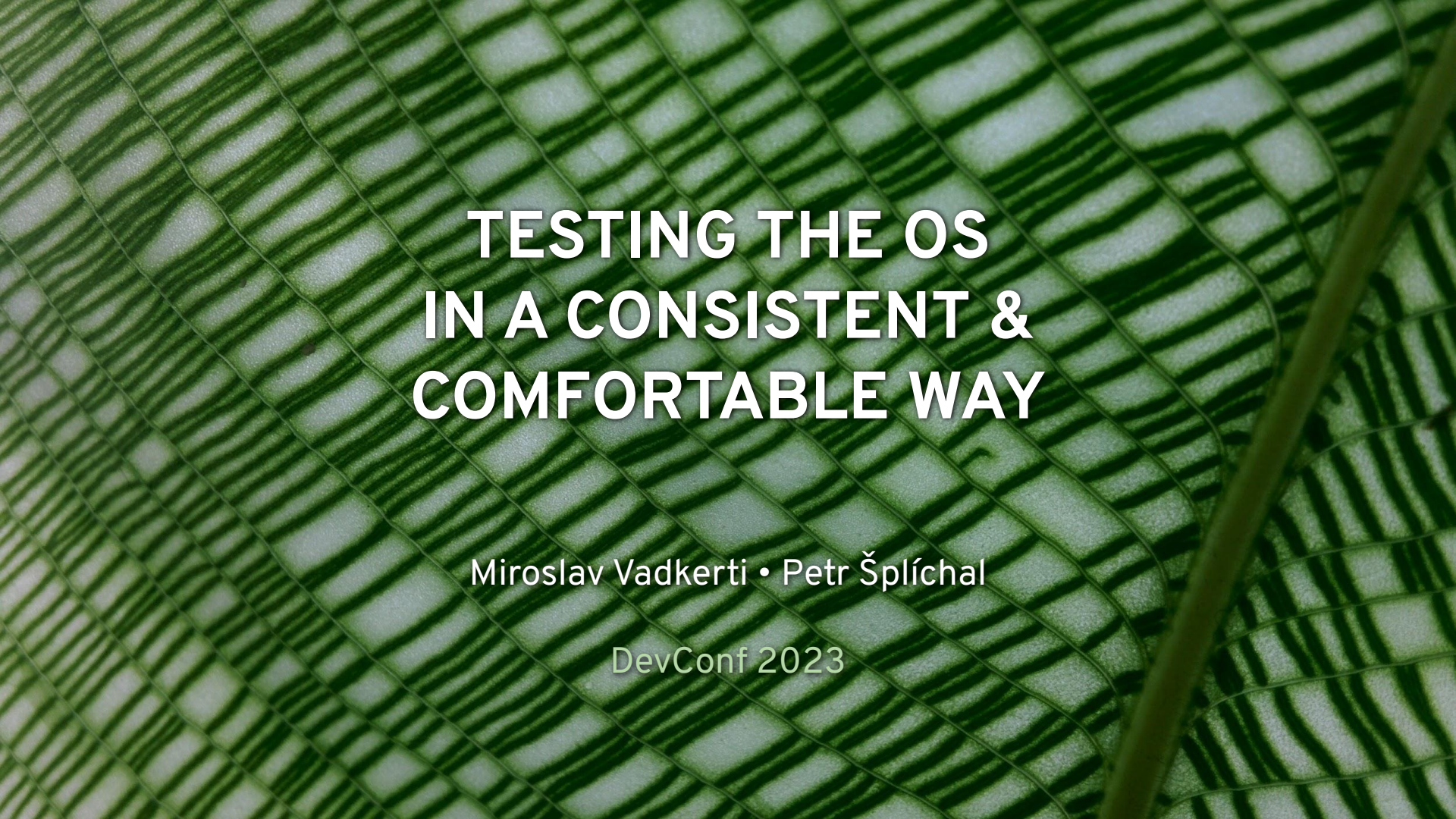
TESTING THE OS
IN A CONSISTENT &
COMFORTABLE WAY
Miroslav Vadkerti • Petr Šplíchal
DevConf 2023


introduction

# WE

Who are we?

- Miroslav Vadkerti
  - thrix / mvadkert@redhat.com
- Petr Šplíchal
  - psss / psplicha@redhat.com

# YOU

Who are you?

- Any Fedora packagers here?
- Have you used tmt before?
- Heard about the fmf format?
- Already using the Testing Farm?
- Any experience with Packit?

# AGENDA

What's the plan?

- Short introduction about tmt & Testing Farm
- Hands on!
  - Enable tests using tmt on your components/packages
  - Or use the sandbox repo for experimenting
  - Please, ask questions any time
- What's new since the last devconf?
- Plans for the near and far future

# TEST MANAGEMENT TOOL

What is it and why was it created?

- Command line tool
  - User-friendly way to work with tests
- Metadata specification
  - Human-readable, concise config
  - L0 = core (summary, description, tag…)
  - L1 = tests (test, require, duration…)
  - L2 = plans (group tests, define environment)
  - L3 = stories (track implementation, test & doc coverage)

# FREEDOM FOR TESTS

- Tests do not need to rely on the internal infrastructure
- No dependency on a particular test case management system (TCMS)
- Stored inside git repositories, convenient to make open source
- Easier cooperation between upstream and downstream
- Test code can be shared across repositories without duplication
- Integration testing across components made easy

# COMFORT FOR USERS

- All test metadata can be stored inside a single git repository
  - Using the fmf format with inheritance and elasticity
  - Simple, human-readable configuration
- Consistent configuration across
  - Packit GitHub/GitLab, Fedora CI, CentOS Stream, RHEL CI
- Execute tests according to your preferences
  - Virtual machine? Container? A physical server? Local host?
  - Sure, no problem!
- Easier cooperation between developer and QE on test code
  - Easy to reproduce issues revealed by CI

# TESTING FARM

An open-source Testing System as a Service

- A flavor of SaaS with a focus on executing automated tests against VMs, bare-metal machines and containers.
- Deployed in hybrid cloud, can run tests against multiple infrastructure clouds
- Provides a single public HTTP API endpoint: [api.dev.testing-farm.io](api.dev.testing-farm.io)
- Supports hardware requirements, infrastructure agnostic, failover / Artemis
- Multiple environments per request, plans parallelization
- Reproducer steps for tmt to easily investigate test failures


get started

# LINKS

A couple of useful links

- tmt docs
  - tmt.readthedocs.io
  - tmt cheat sheet
  - the guide
- fmf docs
  - fmf.readthedocs.io
- fedora quick start guide
  - docs.fedoraproject.org/en-US/ci/tmt
- packit & testing farm
  - packit.dev/testing-farm

# INSTALL

Install tmt on your laptop. Try core/full package, selected provision plugins.

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

# choose your preferred provision plugins
sudo dnf install -y tmt-provision-virtual
sudo dnf install -y tmt-provision-container

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

See the documentation for other installation options.

# TEST REPOSITORY

Check out the repository to store your tests

- Enable the Packit service for your repo
  - github.com/marketplace/packit-as-a-service
- Add a simple .packit.yaml to your repo
  - packit.dev/docs/configuration

- Or use the workshop sandbox for experimenting
  - github.com/teemtee/workshop

# INITIALIZE

Initialize a metadata tree

```
# Initialize the tree with sample metadata
tmt init --template mini

> tree -a
.
├── .fmf
│   └── version
└── plans
    └── example.fmf
```

# SMOKE

Let's look at the created test and execute it in different environments

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


# Run it locally
tmt run --all provision --how local

# Run it in a VM (requires tmt-provision-virtual subpackage)
tmt run
tmt run --all provision --how virtual --image fedora-34
```

# MULTIPLE TESTS

Execute multiple tests inside a single plan using discover step

```
# create a new test using the beakerlib shell framework
tmt test create --template beakerlib /tests/smoke

# example test metadata
summary: A simple test
test: ./test.sh
framework: beakerlib

# create a new plan using the base template
tmt plan create --template base /plans/basic
```

# PLAN STEPS

Separate steps for each test stage, multiple methods

```
discover:
    how: fmf
provision:
    how: container
    image: fedora:fresh
prepare:
    how: install
    package: did
execute:
    how: tmt
```

# DISCOVER

Discover what tests would be run

```
tmt run discover

# select plan
tmt run plan --name upstream discover

# verbose
tmt run plan --name upstream discover -v

# debug
tmt run plan --name upstream discover -vd
tmt run plan --name upstream discover -vdd
tmt run plan --name upstream discover -vddd
```


minimize test maintenance

# CONSISTENT CONFIG

Learn one syntax, use everywhere

## Upstream / GitHub / GitLab

```
summary:
    Run all tier 1-3 tests
discover:
    how: fmf
    filter: "tier: 1,2,3"
prepare:
    how: install
    package: tmt-all
```

## Fedora

```
summary:
    Run integration tests with tmt
discover:
    how: fmf
    url: https://github.com/teemtee/tmt
    filter: 'tier: 1, 2'
prepare:
    how: install
    package: tmt-all
```

## CentOS Stream

```
summary:
    Run integration tests with tmt
discover:
    how: fmf
    url: https://github.com/teemtee/tmt
    filter: 'tier: 1, 2'
prepare:
    how: install
    package: tmt-all
```

## RHEL

```
summary:
    Run integration tests with tmt
discover:
    how: fmf
    url: https://github.com/teemtee/tmt
    filter: 'tier: 1, 2'
prepare:
    how: install
    package: tmt-all
```

# REMOTE DISCOVER

Share test code across repositories

```
# Fetch all tests from the default branch
discover:
    how: fmf
    url: https://github.com/teemtee/tmt

# Select subset of tests from given commit
discover:
    how: fmf
    url: https://github.com/teemtee/tmt
    ref: f524fef
    filter: “tier:1”
```

# MULTIPLE PHASES

Fetch tests from multiple remote repositories

```
discover:
  - name: upstream
    how: fmf
    url: https://github.com/teemtee/tmt
  - name: fedora
    how: fmf
    url: https://src.fedoraproject.org/rpms/tmt/
```

# IMPORT PLAN

Fetch the whole plan config from a remote repository

```
plan:
    import:
        url: https://github.com/teemtee/tests
        name: /plans/polarion
```

# DIST GIT SOURCE

Extract tests from the package sources

- Available for all dist-git repositories
- Patch application not supported yet

```
discover:
    how: fmf
    dist-git-source: true
```

# DYNAMIC REF

Choose the right branch based on the given context

```
discover:
    how: fmf
    url: https://github.com/teemtee/repo
    ref: "@.tmtref"

# dynamic reference rules
ref: main
adjust:
  - when: distro == centos-stream-9
    ref: rhel-9
  - when: distro == fedora
    ref: fedora
```


execute tests easily

# RUN TESTS

Execute tests, choose your preferred environment

```
# run all tests in a vim
tmt run

# run selected plan
tmt run plan --name smoke

# run in a container
tmt run --all plan -n smoke provision --how container
tmt run ... --image fedora:fresh

# run it on your laptop (if you feel safe)
tmt run --all plan -n smoke provision --how local
```

# DEBUG TEST CODE

Fast way to re-execute modified test code

```
# run all steps until test execution
tmt run --until execute

# repeat test execution as needed
tmt run --last execute --force

# log in to the guest to adjust what's needed
tmt run --last login

# apply test code changes, execute again
tmt run --last discover --force execute --force
```

# MIGRATE

Migrate an old Beaker/STI test to tmt

- A couple of examples documented to help with migration from STI
  - How do I migrate STI tests to tmt?
- Easy conversion of old Beaker Makefiles thanks to import:

```
# including tcms integration
tmt test import

# process just the Makefile
tmt test import --no-nitrate
```

# MORE

A couple more useful commands

```
# check run status
tmt status

# clean runs, guests, images
tmt clean

# verify config against the specification
tmt lint

# track implementation, test and docs coverage
tmt stories
```


news & plans

# NEWS

What's new since last year?

- parallel execution & guest topology for multihost testing
- discover: dist-git-source, dynamic ref
- provision: beaker, artemis, improved testcloud
- execute: upgrade, custom results
- report: reportportal, polarion
- finish: workdir pruning
- export: plans, polarion
- import: remote plans, polarion
- other: improved linting, hardware specification extended

# FUTURE

What's ahead of us?

- Next steps
  - Documentation cleanup, extend the guide
  - Continue towards the full Multihost test support
  - Improve test debugging and usability (aliases, wizard mode)
  - Bring tmt reproducer & Testing Farm environment closer to identical
- How to get involved?
  - Submit bugs and ideas for improvement
  - Pick a good first issue and create pull request
  - Join the #tmt channel on Slack to discuss the design


thank you!