Automated Helm Chart Testing with Bats

Fredrik Steen

Fredrik Steen

Director of Software Engineering

Varnish Software

Mail: fredrik.steen@varnish-software.com

LinkedIn: fredriksteen

GitHub: stone

"Blog": https://tty.se/

Linux and open source user/developer since the mid 90s

Psy-trance DJ, Beekeeper, hydroponics, electronics/homeautoumation geek

Agenda Items

- What is Helm and Helm Charts?
- The Good, the Bad and the Ugly (Some ranting)
- Challenges for a product software company
- Why Test Helm Charts?
- Testing Helm Charts
- Bats Crash Course
- Best Practices
- Summary
- Q&A

What is Helm and Helm Charts?

> Helm is a tool for managing Kubernetes applications. Helm uses a packaging format called charts. Charts are collections of pre-configured Kubernetes resources.

- **Helm** is a package manager for Kubernetes
- **Helm** charts are collections of files that describe a related set of Kubernetes resources
- **Helm** manages Kubernetes applications defined as **charts**
- **Charts** can be versioned and shared
- **Charts** are stored in chart repositories
- **Charts** can be installed, upgraded, and deleted using Helm commands

Example of using helm to install Varnish Cache

```
$ vim values.yaml
$ helm install myvarnish oci://registry-1.docker.io/varnish/varnish-cache
Pulled: registry-1.docker.io/varnish/varnish-cache:1.1.1
NAME: myvarnish
LAST DEPLOYED: Fri May 16 10:06:48 2025
STATUS: deployed
REVISION: 1
...

$ helm upgrade -f values.yaml myvarnish oci://registry-1.docker.io/varnish/varnish-cache
Release "myvarnish" has been upgraded. Happy Helming!
NAME: myvarnish
LAST DEPLOYED: Fri May 16 10:07:08 2025
STATUS: deployed
REVISION: 2
...

$ helm rollback myvarnish
$ helm uninstall myvarnish
```

But is it really that good?

The Good

- Package once, deploy anywhere: lets you wrap up a whole app with sensible defaults.
- Customizability (values.yaml): You get one file to rule them all. Change ports? Image tags? TLS? It's all in there.
- Rollback-friendly: Helm tracks releases like Git does commits. One bad deployment? Just roll it back.
- Ecosystem-rich: There's a chart for everything
- Repository support: You can host your own chart repositories or use public ones.
- Community support: Helm has a large and active community, making it easy to find help and resources.

The Bad

- Template hell: Templates are written in Go's templating language, You end up nesting curly braces like this:

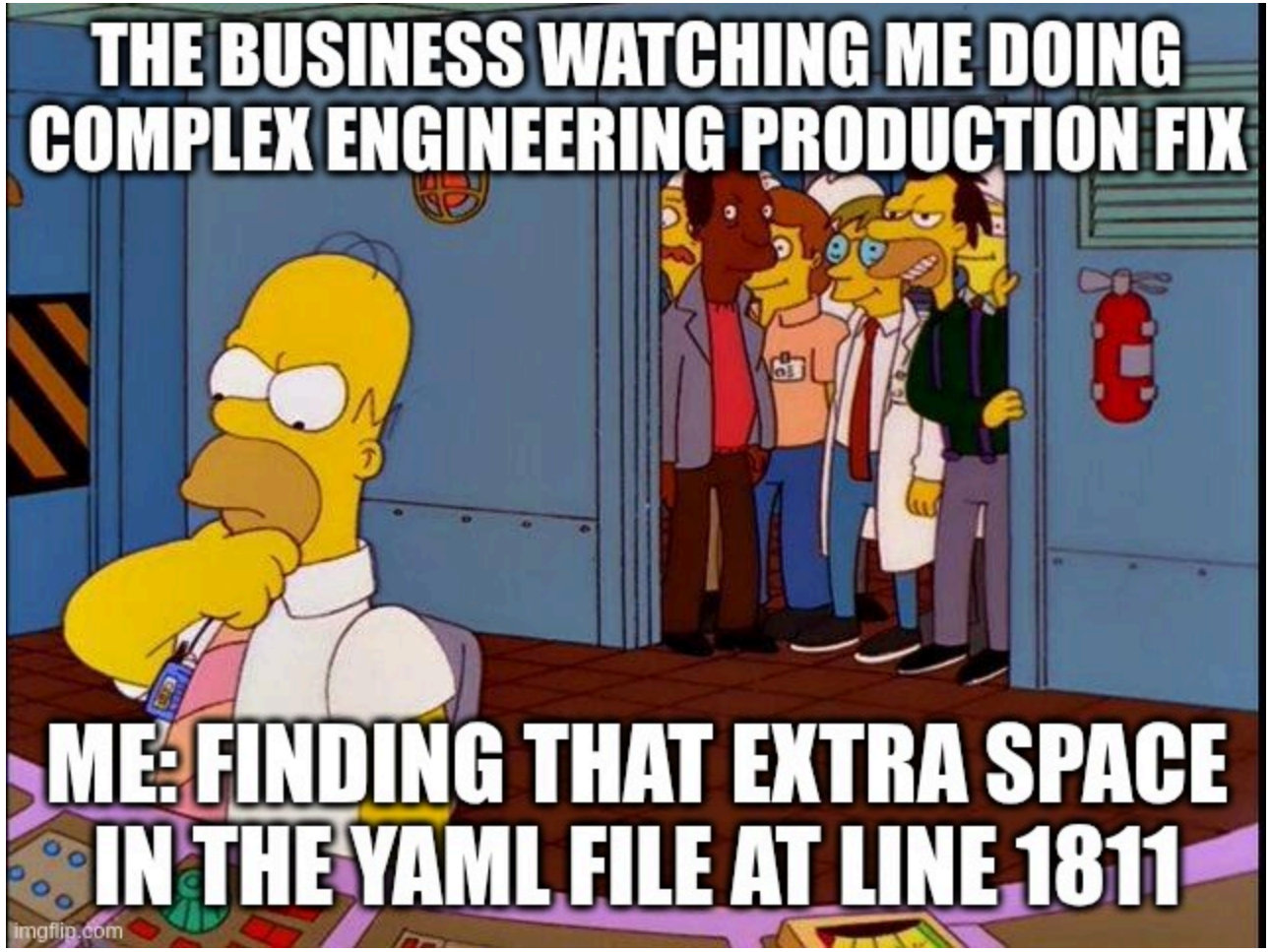
  ```
  {{- if .Values.service.enabled }} port: {{ .Values.service.port }} {{- end }}
  ```

- `values.yaml` becomes a jungle: As your chart grows, values.yaml becomes a sprawling monster.
- Want to change one thing, prepare to wade through 4 layers of defaults, conditionals, and overrides.
- Debugging a Helm chart is like trying to guess what your YAML will look like after rendering.

and the Ugly

- The template engine does not know that it is actually a YAML file
- The template language is dumb, no knowledge about Kubernetes
- Hard to read and understand (parse go templates in your head + indentation hell)

We are stuck with a flawed technology but with a thriving ecosystem



Challenges for a Product Software Company

- Customer environments vary widely in configurations, infrastructure, and dependencies
- Limited visibility into customer environments creates significant complexity in diagnosing and resolving issues
- Version fragmentation across customer base increases support complexity
- Bug fixes in one version can introduce regressions in others
- Need to ensure compatibility and performance across diverse deployment scenarios
- Balancing new feature development with maintenance of existing versions

We need to test our Helm Charts!

- Systematically tests how different `values.yaml` affect deployed resources, ensuring customization options work as expected
- Catches breaking changes early before they reach customer environments, especially critical for customer-hosted deployments
- Tests run in CI/CD pipelines for continuous testing
- Distribution of charts to end-users **requires confidence** in their quality

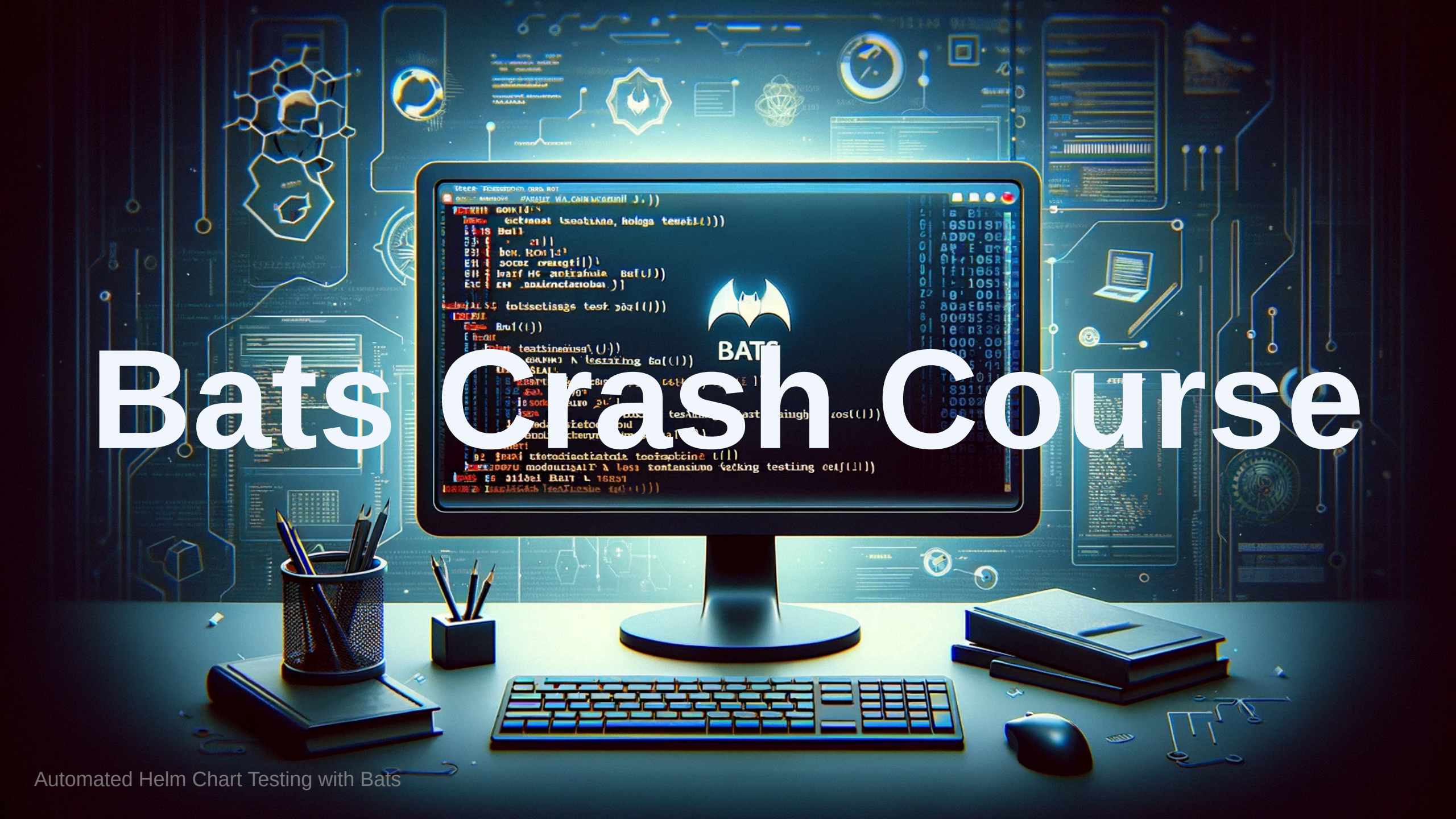
Testing using Bats

> Bash Automated Testing System

- Bats is a testing framework for Bash
- Free and Open Source, been around for +10 years
- Tests written in straightforward Bash syntax
- Can be used for unit tests, integration tests, and end-to-end tests
- Can be used for testing almost anything that can be run in a shell
- TAP-compliant (Test Anything Protocol), JUnit XML, and JSON output formats

Bats Test Structure

- `.bats` files contain one or more `@test` blocks
- Setup and teardown functions for environment management
- `run` command to execute commands and capture Output
- Assertions for checking command status, output, and error messages
- `load` command to include helper functions
- Mock functions for simulating external commands
- Support for tags, pre/post hooks, and test filtering
- All in bash syntax!


Bat$ Crash Course
BATS
Automated Helm Chart Testing with Bats

We want to test this script: `greet.sh`

```
#!/usr/bin/env bash
echo "Hello, $1!"
```

Tests for `greet.sh`, using bats.

```
#!/usr/bin/env bats

@test "greet function" {
    run bash greet.sh "Karlstad"
    [ "$status" -eq 0 ]
    [ "$output" = "Hello, Karlstad!" ]
}

@test "greet function without name" {
    run bash greet.sh
  [ "$status" -eq 0 ]
  [ "$output" = "Hello, !" ]
}
```

Running the tests:

Regular "pretty" output:

```
[stone@stout crash-course]$ bats test_greet.bats
test_greet.bats
 ✓ greet function
 ✓ greet function without name

2 tests, 0 failures

[stone@stout crash-course]$
```

Junit output:

```
[stone@stout crash-course]$ bats -T -F junit test_greet.bats
<?xml version="1.0" encoding="UTF-8"?>
<testsuites time="0.010">
<testsuite name="test_greet.bats" tests="2" failures="0" errors="0" skipped="0" time="0.010" timestamp="2025-05-14T08:01:45" hostname="stout">
    <testcase classname="test_greet.bats" name="greet function" time="0.005" />
    <testcase classname="test_greet.bats" name="greet function without name" time="0.005" />

</testsuite>
</testsuites>
```

Example 1 — Unit Testing Helm Templates

> Verify the existence of a deployment template

```
#!/usr/bin/env bats

@test "Deployment: enabled by default" {
    run helm template \
        --namespace default \
        --show-only templates/deployment.yaml \
        .
    [ "$status" -eq 0 ]
    [ "$(echo "$output" | yq -r 'length > 0')" = "true" ]
}
```

Example 2 — Disabling a Deployment

> Verify that the deployment is disabled when `server.kind` is set to `DaemonSet`

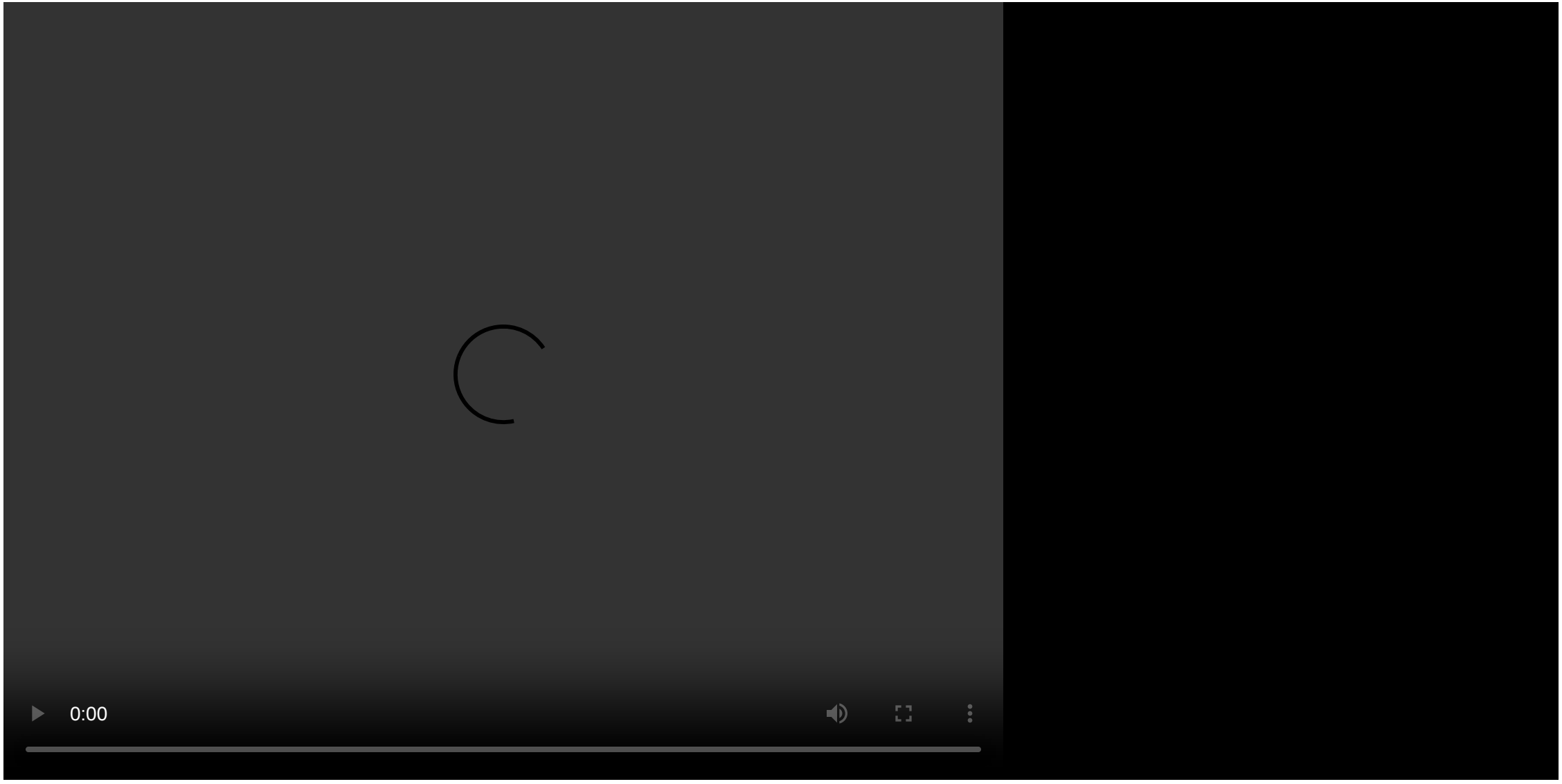
```
#!/usr/bin/env bats

@test "Deployment: can be disabled" {
    run helm template \
        --set 'server.kind=DaemonSet' \
        --namespace default \
        --show-only templates/deployment.yaml \
        .
    [ "$status" -ne 0 ] || [ "$(echo "$output" | yq -r 'length > 0')" = "false" ]
}
```

Example 3 — E2E Testing with Kubernetes

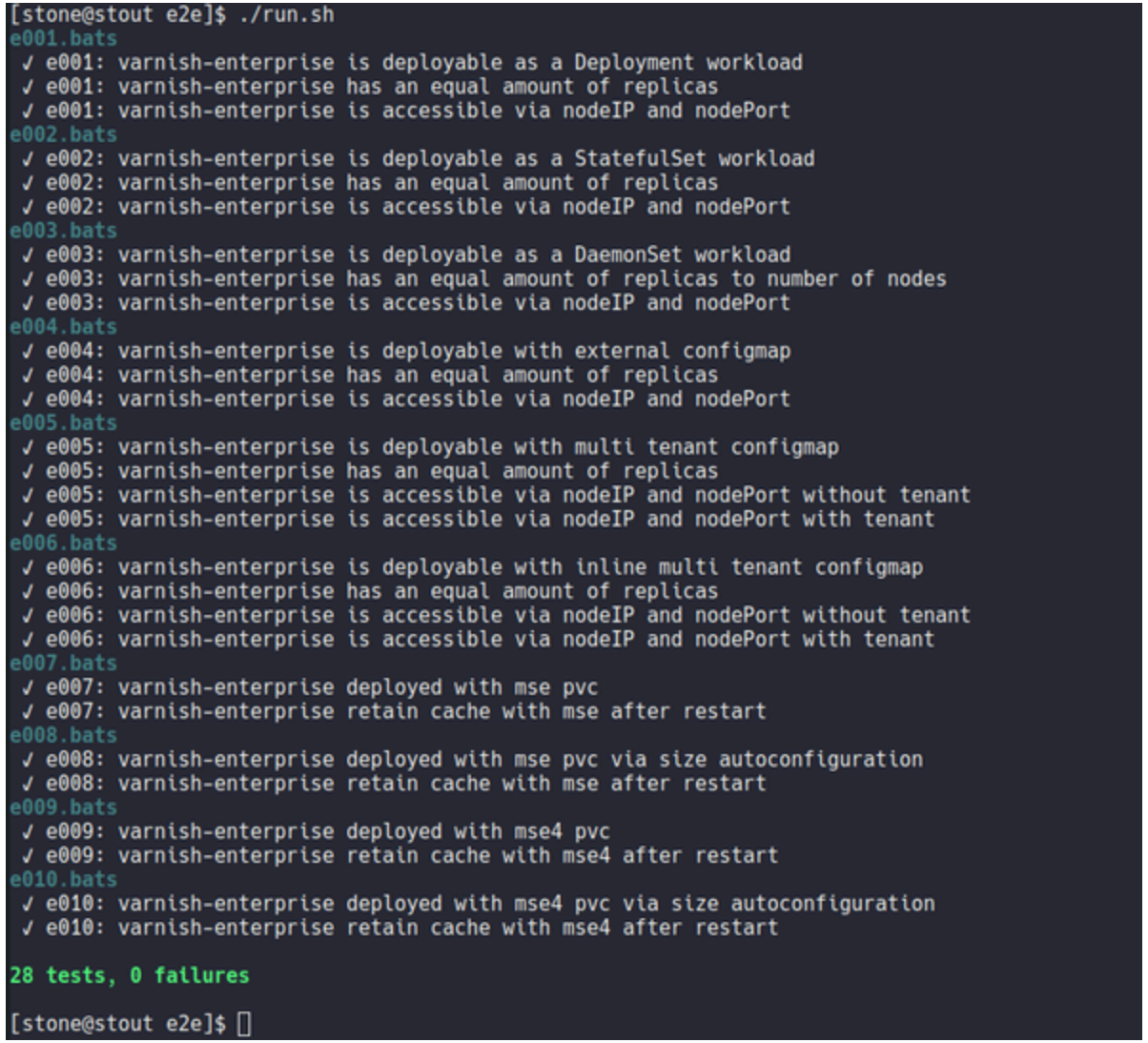
```
#!/usr/bin/env bats
load _helpers

setup_file() {
  install_namespace
  install_pull_secret
  kubectl apply -f deployment-counting-service.yaml
  kubectl apply -f service-counting-service.yaml
  helm install varnish-enterprise
  kubectl rollout status deployment varnish-enterprise --timeout=60s
}

@test "e001: varnish-enterprise is deployable as a Deployment workload" {
  run kubectl get deployment -l app.kubernetes.io/name=varnish-enterprise -o json
  [ "$status" -eq 0 ]
  [ "$(echo "$output" | jq -r '.items | length')" -eq 1 ]
}
```

Demo

1/2 Pre-recorded backup slide if demo fails ;)



2/2 Pre-recorded backup slide if demo fails ;)



Best Practices

Structure Your Tests Logically

- Organize by chart component (deployment, service, configmap)
- Use consistent naming conventions for test files
- Group related tests within the same test file

Best Practices

Create Helper Functions

- Build a library of reusable test helpers
- Abstract common operations like chart directory location
- Implement functions for template rendering and validation

Best Practices

Validate Value Propagation

- Ensure values from values.yaml propagate correctly
- Test value overrides work as expected
- Verify default values are applied when not specified

Best Practices

Simplify Commands

- Avoid complex command chains when possible
- Use built-in tools instead of custom workarounds
- Prefer direct assertions over intermediate variables
- For kubernetes / helm consider https://github.com/bats-core/bats-detik

Best Practices

- Use `yq` / `jq` for YAML/JSON parsing
- Clean up resources after tests
- k3d for local Kubernetes testing
- Constantly add new tests even for fixed bugs
- Use CI/CD pipelines for automated Testing
- Run tests in Docker containers for isolation

Alternatives

- Helm Chart tests - Helm's built-in testing framework
- terratest - A Go library for writing tests for infrastructure code
- Conftest - A tool for writing tests using Open Policy Agent (OPA) (Rego language)

Summary

- Bats makes Helm chart testing "easy" and scriptable
- Bats tests are easy to read and maintain
- Bats can be used for much more than just Helm charts
- Supports both template and live cluster validation
- Increases confidence and reduces production issues

Q&A

- Thank you for your attention!
- Questions?