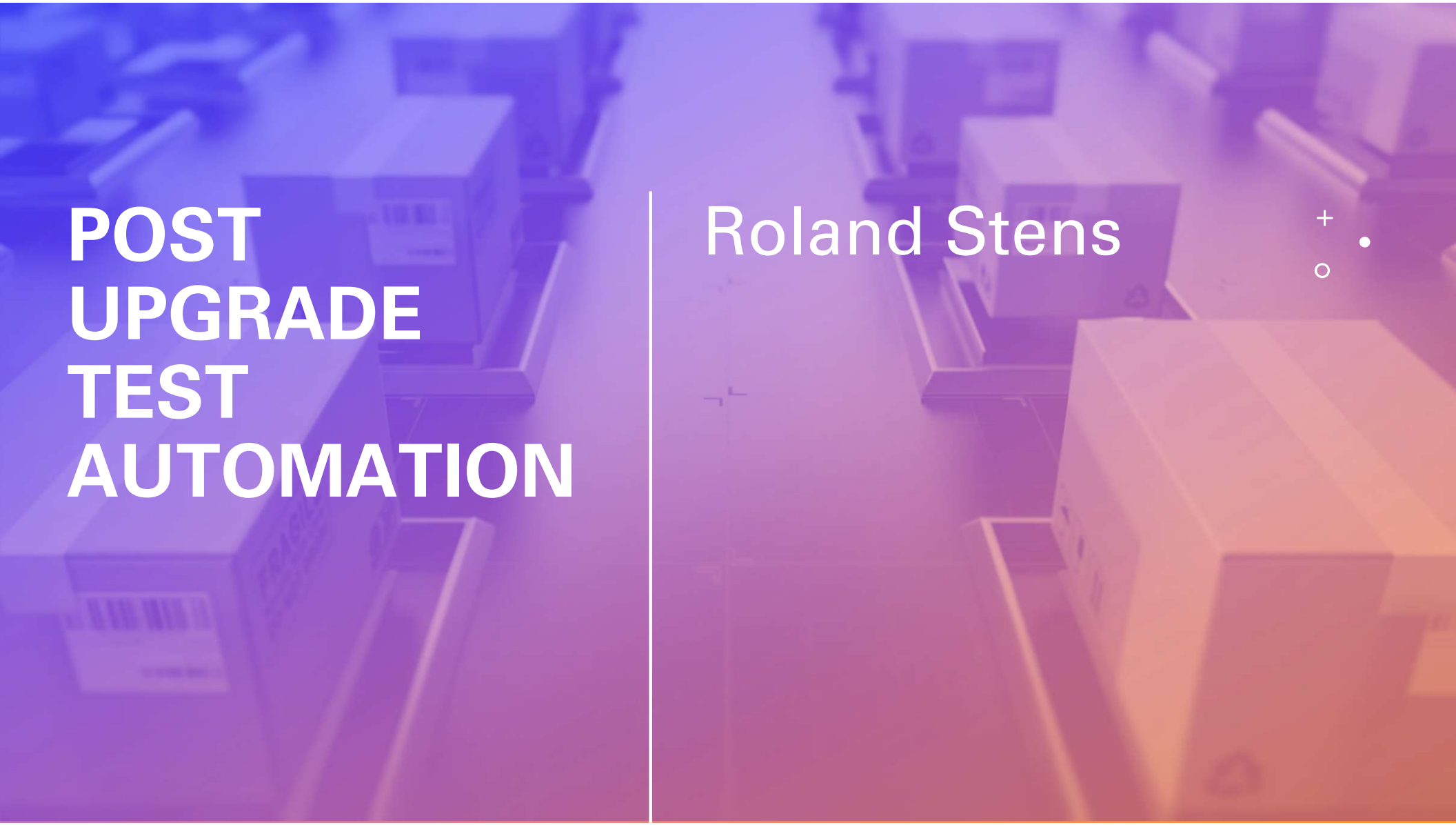


POST UPGRADE TEST AUTOMATION

Roland Stens

Scope of Work

- Build and implement a **modular, extensible automation test framework** for the Platform Services team and add 3 test cases to it.
- Our preference is to include a test case for
  - **Patroni database** clusters that covers testing the general availability of a database cluster and the failover within the cluster,
  - a test case for checking availability of the **Vault** Secret Management platform service that implements Hashicorp Vault product,
  - and a test case for checking availability of the **Artifactory** Trusted Repository service that implements JFrog Artifactory product.

Scope of Work – In short

- A modular, extensible automation test
- Test cases for
  - Patroni database
  - Vault
  - Artifactory
- Tests for:
  - Availability
  - Key functionality
- Desired Result: Obtain/Report status for above applications post-upgrade

Approach Intro

- We need to think about a *framework* which can be utilized to not only meet current demands but also future growth and expansion
- Such a framework needs to support:
  1. Recognize Trigger events (schedule, post update, hardware changes, manual start etc.)
  2. Defining tests/checks (against platform, pod, API, UI etc.)
  3. Running tests/checks (On OpenShift, GitHub Actions, Local)
  4. Triggering appropriate actions when issues are detected
  5. Reporting results

Approach: Trigger Events

- We recognize the following events as triggers for running the tests/checks:
  - Changes/Updates to platform (SW/HW)
  - Environmental Changes – Network, Security, Failover location etc.,
  - Changes/Updates to application
  - Manual triggers (kick off on demand)
  - Scheduled triggers (cron)

Approach: Test and Checks

- Common Health Checks (on Pod)
- Configuration Scans (check for Kubernetes API versions)
- System Usage Scans (Delta of before and after update usage)
- API Functionality (Create, Read, Update, Delete, Version Check)
- Application Checks through the UI (Simulating actual users)
- Application checks directly on the running pod (e.g., issuing a SQL command or running a bash script)

Approach: Action/Workflow

- Unique to this framework is the need to think about actions after issues are detected
  - Automatic Issue reporting in ZenHub/Jira
  - Notifications
    - SMS
    - Email
    - RocketChat
    - Other escalations
  - Automatic take down/restart of failing applications like
    - Hanging pods
    - Pods using too many resources
  - Manual Action

Approach: Result Reporting

- Email/SMS
- Notifications to RocketChat or similar tools
- Dashboard (Grafana)
- Log files/History (stored on S3)

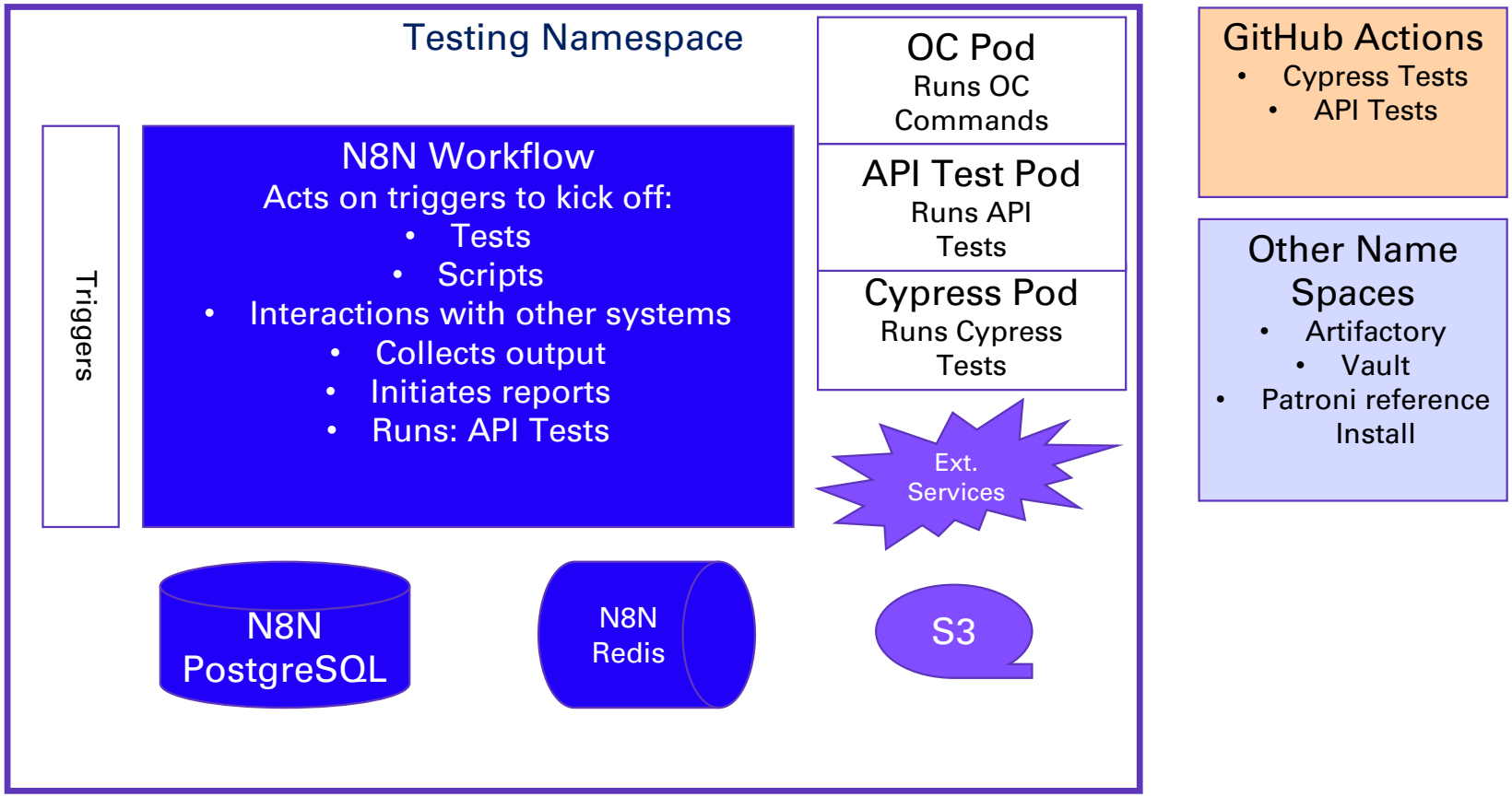
Approach

- Details in: https://docs.google.com/document/d/1RmYw_vKzVKykyY6d3NUSuvGYc3NLYjPS/edit?usp=sharing&ouid=107922000815918263241&rtpof=true&sd=true

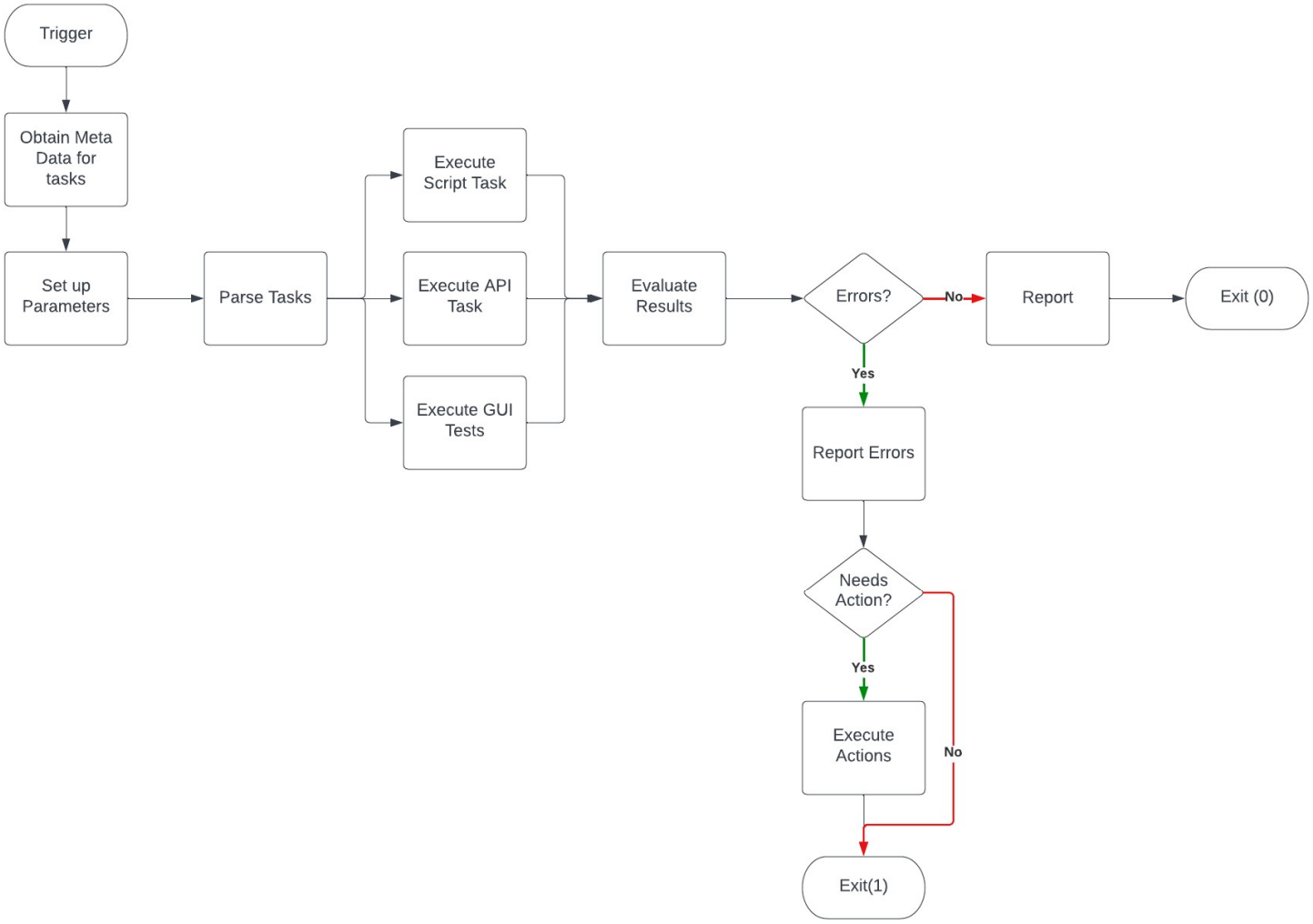
Solution Strategy

- **Conductor/Workflow engine**: N8N (https://n8n.io/)
- **Command line tools** for the platform like: oc, common range of Linux tools, bash etc.
- **Web UI Testing tool**: Cypress https://cypress.io/
- **API Testing Tool**: Cypress, Postman or N8N
- **OpenShift API**: Check status of objects running in OpenShift
- **GitHub Actions** for external checking
- **Email/SMS** Notification services
- Current **reporting and monitoring tools** (e.g. Grafana)
- Several **pods** in a **dedicated Openshift name space**
- **User IDs** with the appropriate access for the different purposes (dba access, admin access, user access etc.)
- **Test installation of Patroni** and/or other tools that cannot otherwise be tested (thinking about failover testing)

Architecture Picture



Post Upgrade Test Flow



Demo/Overview

- https://docs.n8n.io/

Initial Scope of Work

- Acquire NameSpace and resources
- Acquire needed access and authorization
- Build N8N image with needed tools (oc, Cypress etc.)
- Deploy N8N
- Build Trigger/Cron
- Build first Health Check for first application (Artifactory?)
- Build API (version) check for first application
- Build Login for first Application (Cypress)
- Build log write out to S3
- Build email notification

Challenges

- Specific functionality like Patroni failover triggering
- Detecting of application state when no API is available
- Authentication, as methods might differ between applications
- Cross Platform Access
- Chicken and Egg problem
- Etc.

Future

- Adding new applications
- Opening this as a service to the community
- ...