Shifting Performance Engineering Left

Building Performance into the Development Lifecycle

Joe Talerico
Perf&Scale Engineering

José Castillo Lema
Perf&Scale Engineering

Red Hat

whoarewe

- **Joe Talerico** - 13 years in the Performance Team
- Worked on virt, SPEC, OpenStack, and OpenShift

- 6 years in Red Hat
- 4 years in perf/scale department
- 2 years as Telco Cloud Architect in the Solutions & Technology Practices team

Red Hat

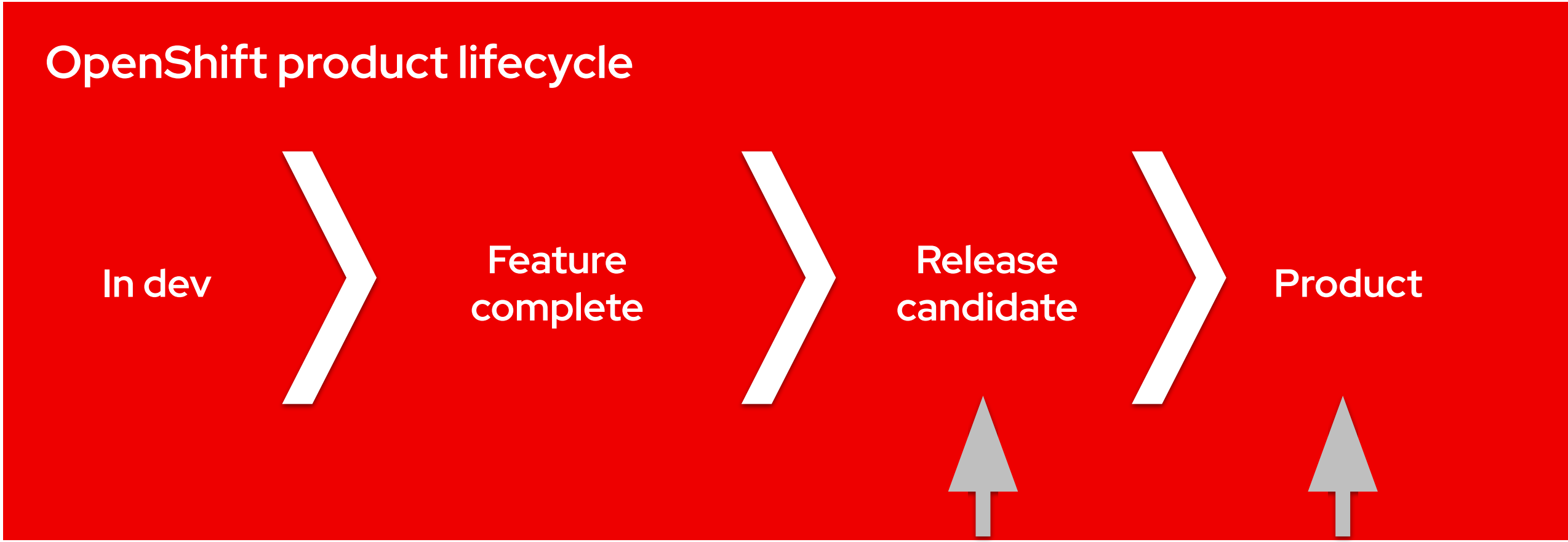
Agenda

1. The problem
2. Shift-left engineering
3. Continuous Performance Testing (CPT)
4. Workflow
5. Tooling
6. Real life case study
7. Challenges and next steps

The problem

- Performance testing often occurs **too late in the pipeline**
- Late detection of performance issues increases **cost**
- Real-world consequences: downtime, latency, etc.

Performance Testing – The past



At best we normally would fit in at the end of the development cycle

Red Hat

What is CPT?

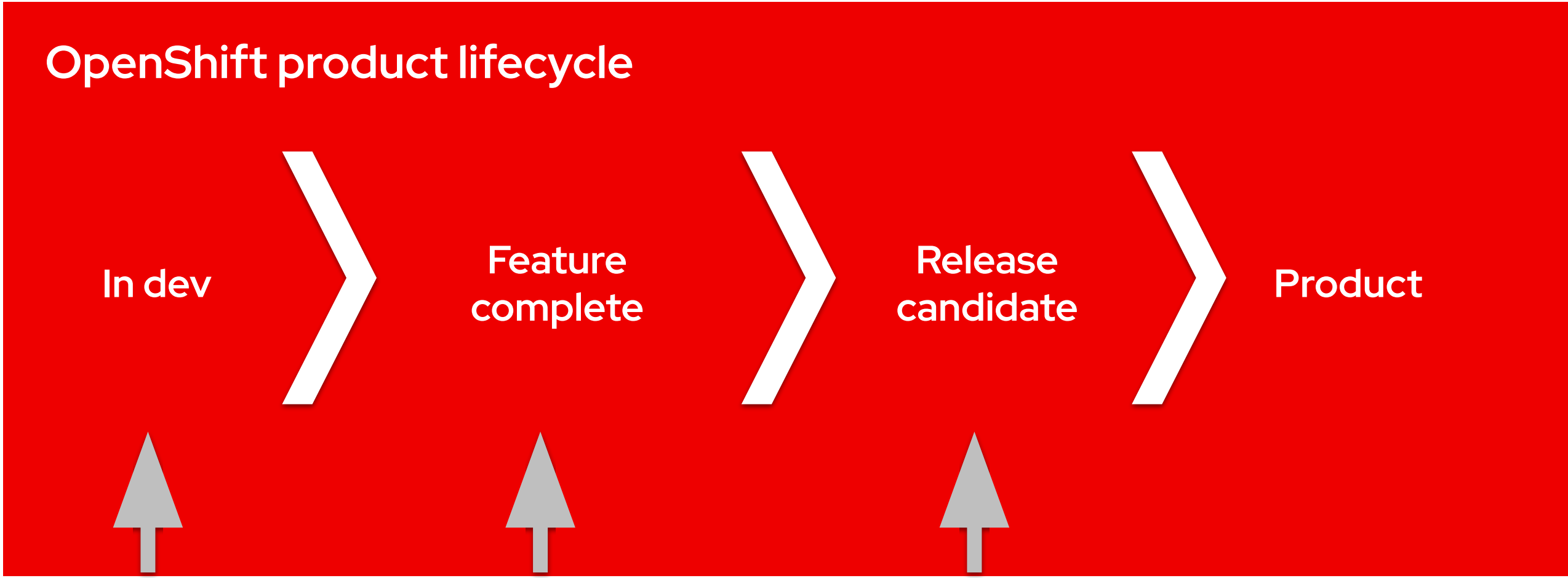
Continuous Performance Testing (CPT) is the practice of executing performance benchmarks (tests) continuously throughout the development lifecycle.

The execution of said Performance benchmarks is **orchestrated in an automated pipeline** (like Jenkins, Prow, etc.), preferably in the pipeline your developers are working out of.

Why CPT?

- Shifts Performance testing left
- Earlier feedback enables faster fixes
- Reduces overall testing costs
- Improves developer ownership of performance → Continuous Improvement Mindset
- More time for technical improvements
- Allows us to drastically improve the coverage matrix and frequency of the testing
  - More data → Need for automatic **performance regression frameworks**
    - **Change point detection** mechanisms

CPT – goals



Where we should fit in the development cycle

Red Hat

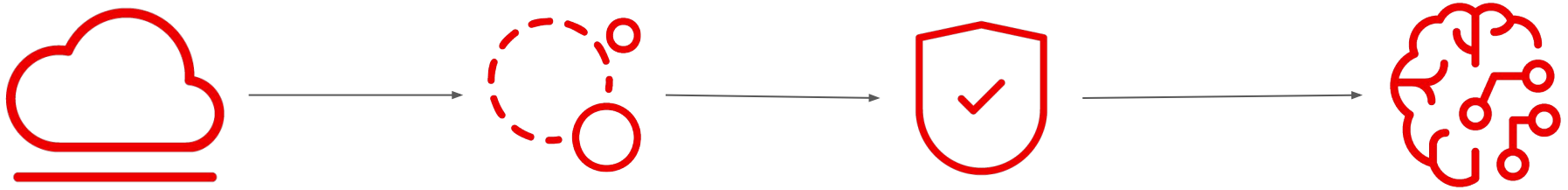
CPT – Cultural shift

- From siloed to **dev + Performance collaboration**
- Embed performance mindset in development sprints
- Performance as part of 'Definition of Done'

Key principles to **navigate change**:

- Document workflows and KPI’s
- Collaboration with QE counterparts is key
  - Handoffs
  - Responsibility matrix

CPT workflow



Deploy product

Run perf test

Store results

Perf regression analysis

Red Hat

Achievements

- Catch performance/scalability regressions in an early stage
- Redefined a new test/platform coverage matrix standard

5 platforms

AWS, Azure, GCP, IBM, Baremetal +
Managed services offerings: ROSA and ARO

+100 weekly tests

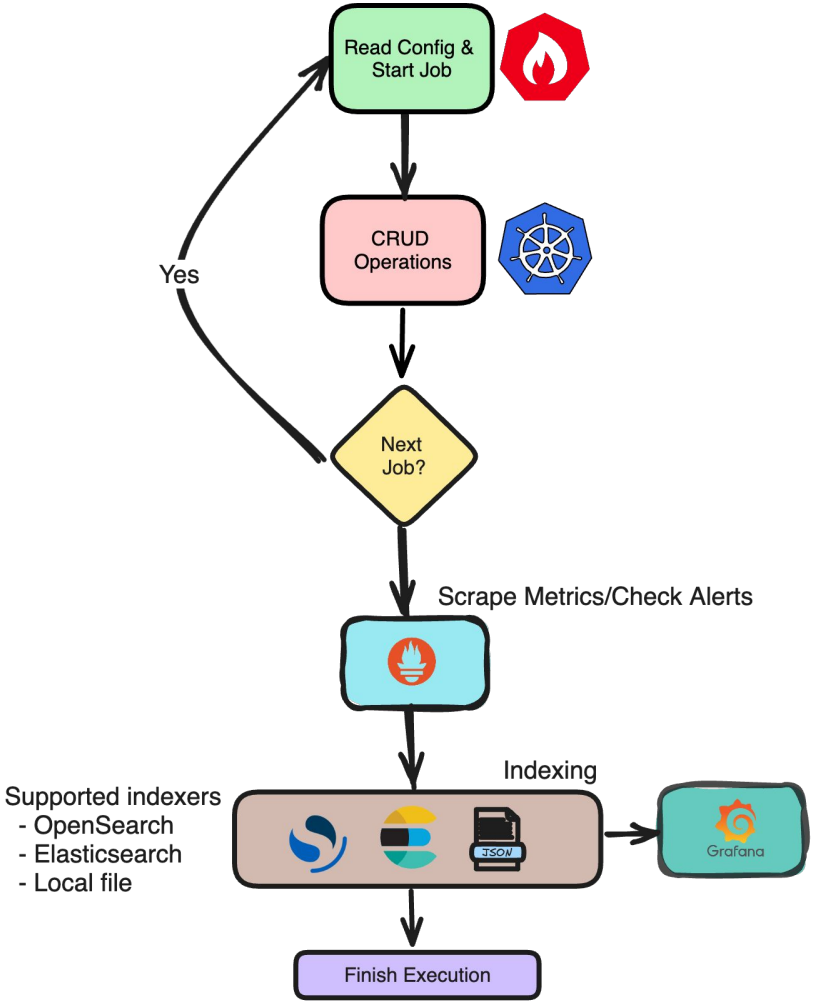
Covering nightly, early candidates, release candidates, stable releases and long term support

Platform +

- OpenShift Virtualization
- Layered products (OLS, ACS, Kepler, ...)

Example Tooling

kube-burner



SCAN ME

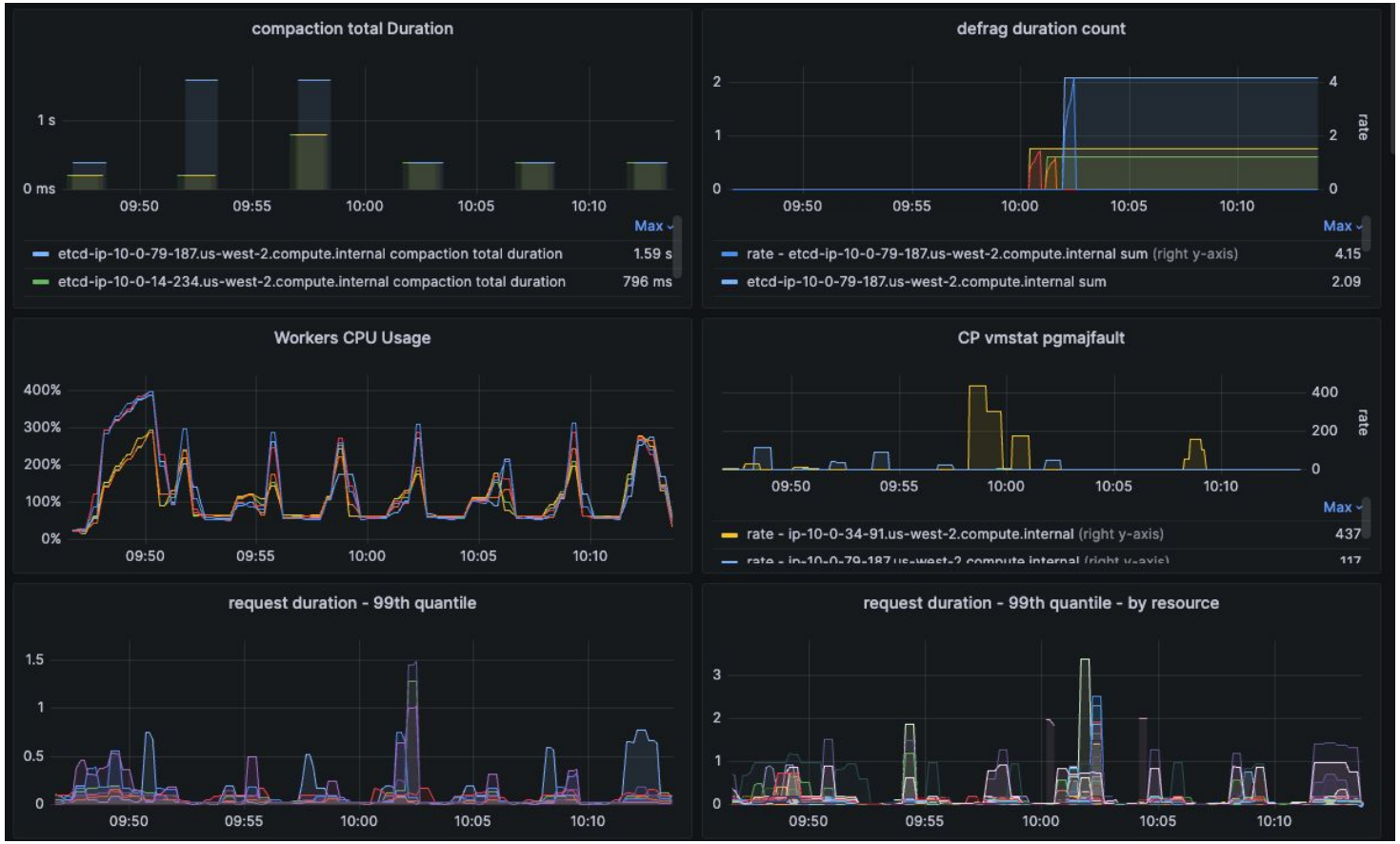
Red Hat

Real life case study

time	uuid	buildUrl	readOnlyAPICallsLatency_max
2025-01-13 11:38:02 +0000	a485ef5b-..-e6d6b35532ca	https://prow...aws-4.19-..payload-control-plane-6nodes/1878744929488867328	0.984
2025-01-14 19:20:40 +0000	95446d27-..-f2d8d7d8ce27	https://prow...aws-4.19-..payload-control-plane-6nodes/1879221652647055360	0.984
2025-01-15 03:28:01 +0000	cc0b1a6d-..-68a59992b6b8	https://prow...aws-4.19-..payload-control-plane-6nodes/1879343743165796352	0.984
2025-01-15 07:43:23 +0000	20058cc2-..-7009402a6c13	https://prow...aws-4.19-..payload-control-plane-6nodes/1879411213251645440	0.976
2025-01-16 01:26:21 +0000	cdf21278-..-71782e3ab17e	https://prow...aws-4.19-..payload-control-plane-6nodes/1879675626953117696	0.984
2025-01-16 08:26:51 +0000	0b4d30e2-..-e056c094a622	https://prow...aws-4.19-..payload-control-plane-6nodes/1879784780980031488	0.984
2025-01-16 16:23:04 +0000	bfb0efa0-..-d1b0cda6dd41	https://prow...aws-4.19-..payload-control-plane-6nodes/1879902896481374208	0.9
2025-01-17 00:08:31 +0000	2d6b598f-..-64d5c98e4236	https://prow...aws-4.19-..payload-control-plane-6nodes/1880020138195947520	3.98
			+923.1% !!
2025-01-17 06:37:24 +0000	391817f4-..-557cd13bd32e	https://prow...aws-4.19-..payload-control-plane-6nodes/1880118891967942656	19.9
2025-01-17 11:39:51 +0000	db926aaf-..-da59c974574a	https://prow...aws-4.19-..payload-control-plane-6nodes/1880193587291885568	19.9
2025-01-17 18:23:02 +0000	5948ce33-..-94cc8cb517fb	https://prow...aws-4.19-..payload-control-plane-6nodes/1880291350289584128	3.94
2025-01-18 00:51:31 +0000	efa220e8-..-60e4876a3e52	https://prow...aws-4.19-..payload-control-plane-6nodes/1880388538206261248	19.6
2025-01-18 06:12:03 +0000	57aff787-..-28e24cf051f7	https://prow...aws-4.19-..payload-control-plane-6nodes/1880474335618011136	28.6
2025-01-18 12:57:00 +0000	11e53cf9-..-9f77ebfc9bdc	https://prow...aws-4.19-..payload-control-plane-6nodes/1880574934015545344	3.86
2025-01-18 18:16:42 +0000	82069eee-..-b3d06c2a1fa8	https://prow...aws-4.19-..payload-control-plane-6nodes/1880655151874707456	3.74

Red Hat

Real life case study



Red Hat

Real life case study

- Look at changelog for the version in question
- Make a guess (RHCOS) at which OpenShift component changed
  - None of the other changes looked suspicious
- Confirm whether this change alone causes the issue
- Comparing RHCOS **good** (kernel 427) vs **bad** (kernel 570)
  - Separated by 143 builds
  - Forked in their own Z-streams (e.g. el9_4 and el9_6)
- Guess again: Pick the kernel

Real life case study

- Focus on one single build containing 618 commits
- Kernel team took bisection down through all commits to land on a single (or two) change(s)

The **madvise**() system call is used to give advice or directions to the kernel about the address range beginning at address *addr* and with size *size*. **madvise**() only operates on whole pages, therefore *addr* must be page-aligned. The value of *size* is rounded up to a multiple of page size. In most cases, the goal of such advice is to improve system or application performance.

MADV_RANDOM
Expect page references in random order. (Hence, read ahead may be less useful than normally.)

Challenges

- Resistance to change and lack of expertise
- Tooling integration issues
- We can only detect regressions on metrics we are tracking
- Avoid **duplicated testing**
  - Reduce redundant tested combinations
- Enable developers to **interpret** (and predict) performance results
  - Exploring the use of AI to help* analyze results

Solutions

Key principles to **navigate change**:

- Training
- Automation
- Establish a solid protocol for dealing with test failures and flakiness
- Champion roles

Roadmap – where to start

- Start with one service
- Educate teams on performance fundamentals
- Build integrated performance dashboards into CI/CD pipelines
- Include performance in code reviews

Food for thought:

- How can performance testing fit into your current sprint workflow?
- What's one thing you could shift left today?

Wrap up

1. Shift-left engineering
2. Continuous Performance Testing (CPT)
3. Workflow
4. Tooling
    a. kube-burner
5. Real life case study
    a. OCP → CoreOS → Kernel → syscall
6. Challenges and next steps

Red Hat Summit

Thank you

 linkedin.com/company/red-hat

 facebook.com/redhatinc

 youtube.com/user/RedHatVideos

 twitter.com/RedHat

Red Hat