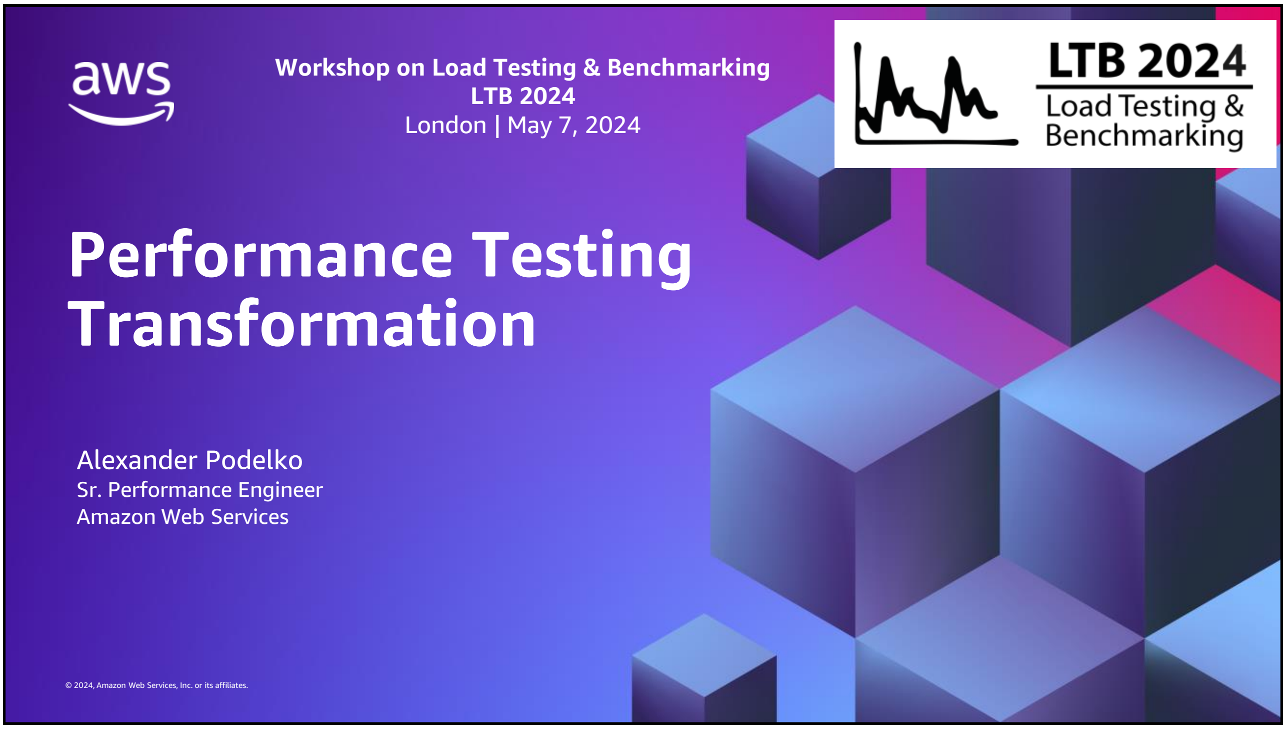


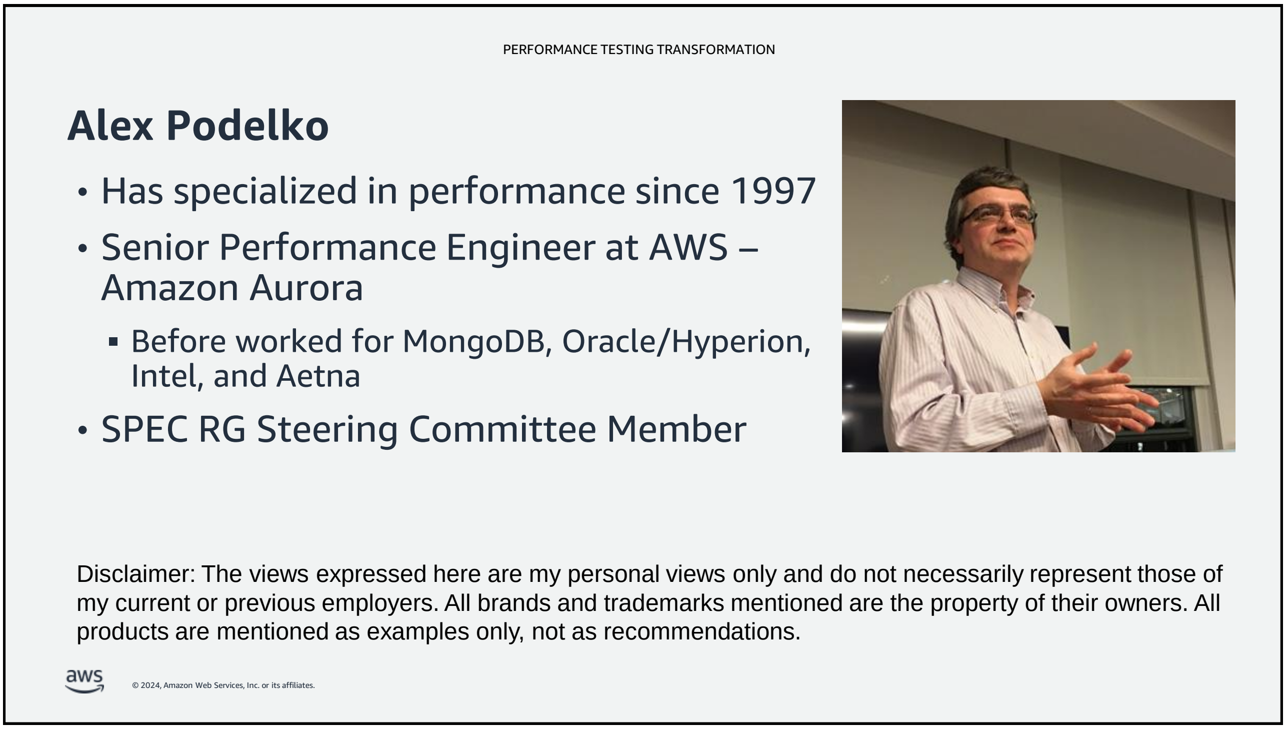
Workshop on Load Testing & Benchmarking
LTB 2024
London | May 7, 2024

LTB 2024
Load Testing & Benchmarking

Performance Testing Transformation

Alexander Podelko
Sr. Performance Engineer
Amazon Web Services

1



PERFORMANCE TESTING TRANSFORMATION

Alex Podelko

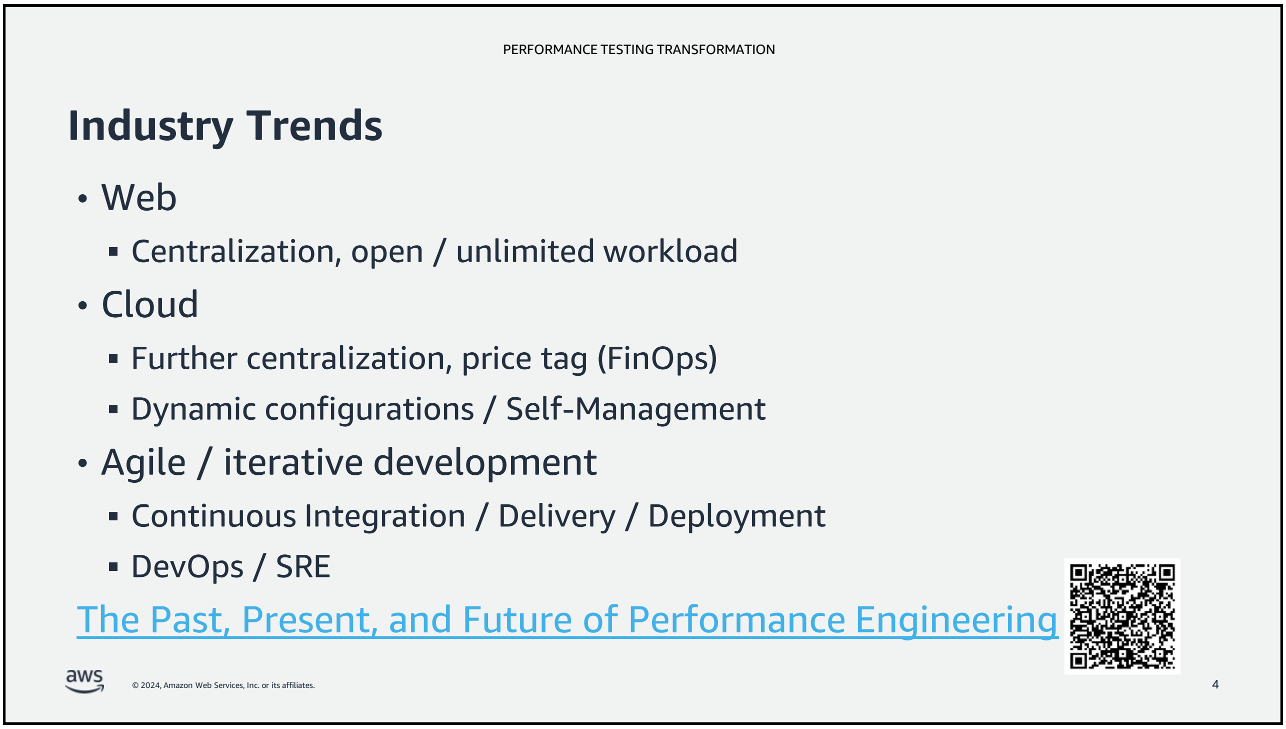
- Has specialized in performance since 1997
- Senior Performance Engineer at AWS – Amazon Aurora
  - Before worked for MongoDB, Oracle/Hyperion, Intel, and Aetna
- SPEC RG Steering Committee Member

Disclaimer: The views expressed here are my personal views only and do not necessarily represent those of my current or previous employers. All brands and trademarks mentioned are the property of their owners. All products are mentioned as examples only, not as recommendations.

2

Adjusting Performance Testing to Industry Trends:

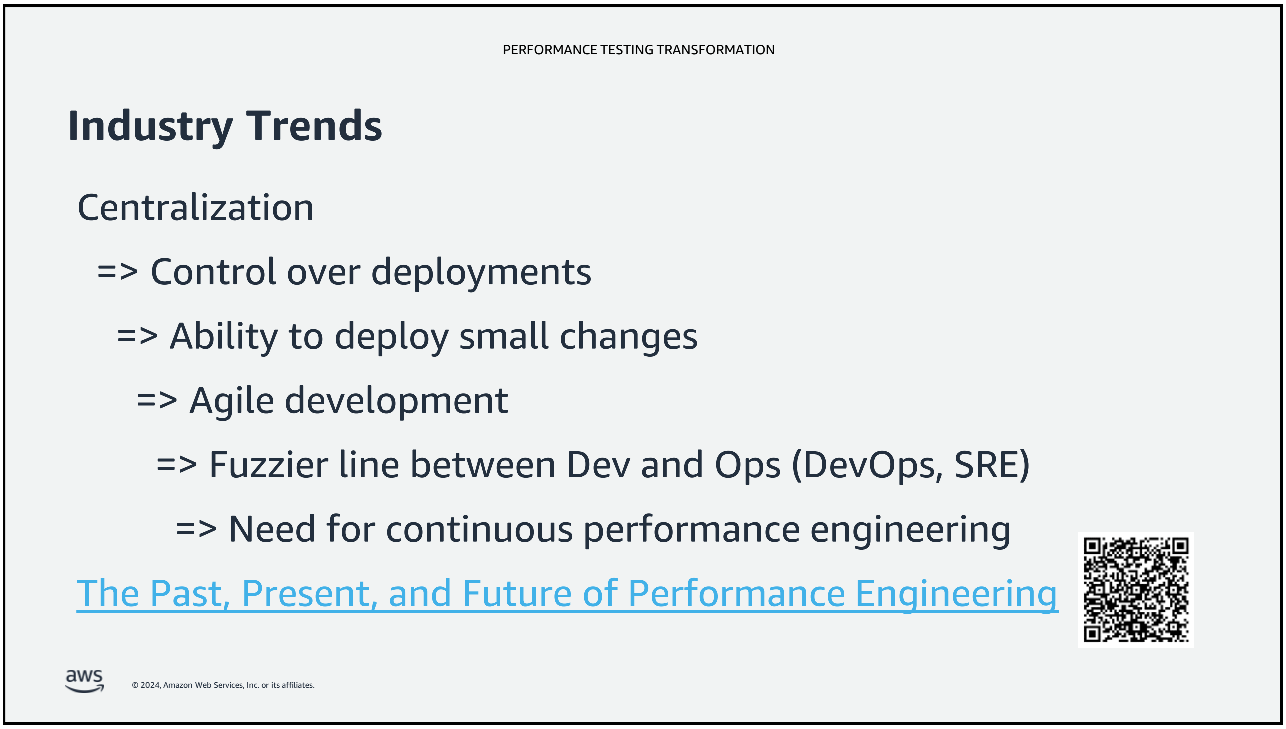
<u>Adding</u> early and continuous performance testing



Industry Trends

- Web
  - Centralization, open / unlimited workload
- Cloud
  - Further centralization, price tag (FinOps)
  - Dynamic configurations / Self-Management
- Agile / iterative development
  - Continuous Integration / Delivery / Deployment
  - DevOps / SRE

[The Past, Present, and Future of Performance Engineering]



Industry Trends

Centralization

=> Control over deployments

=> Ability to deploy small changes

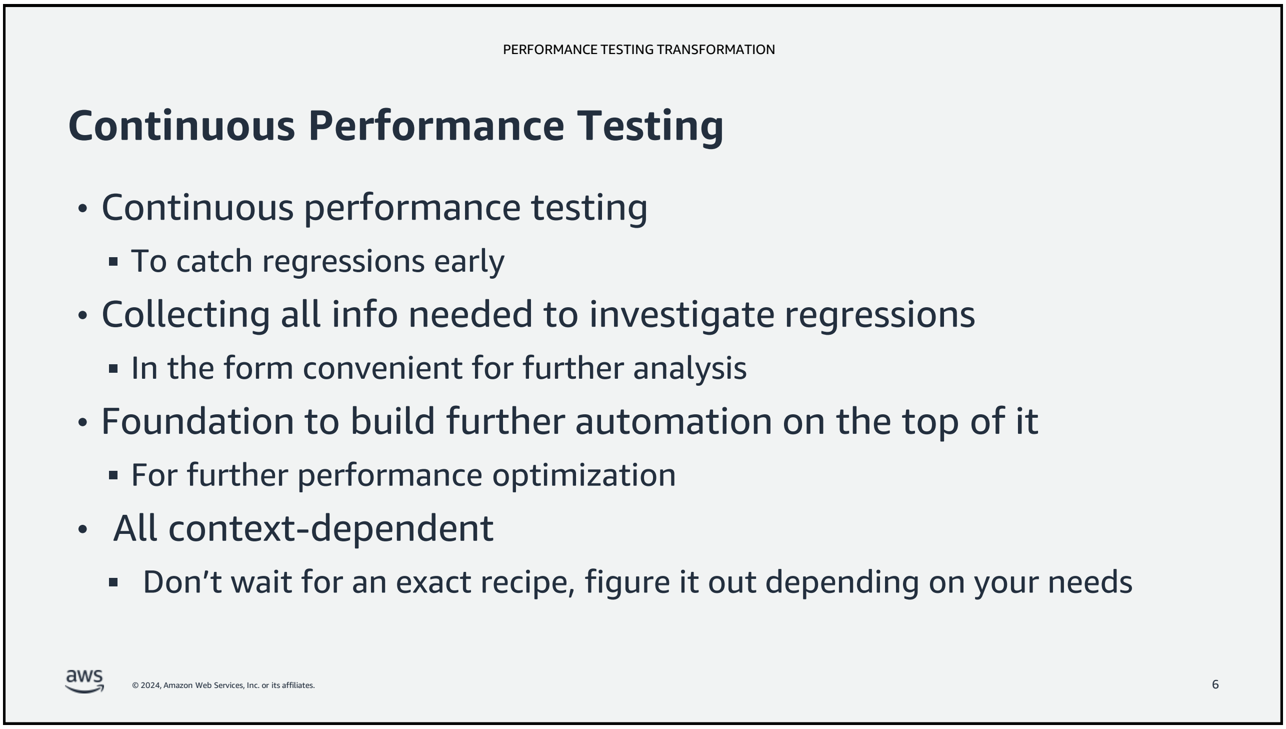
=> Agile development

=> Fuzzier line between Dev and Ops (DevOps, SRE)

=> Need for continuous performance engineering

The Past, Present, and Future of Performance Engineering

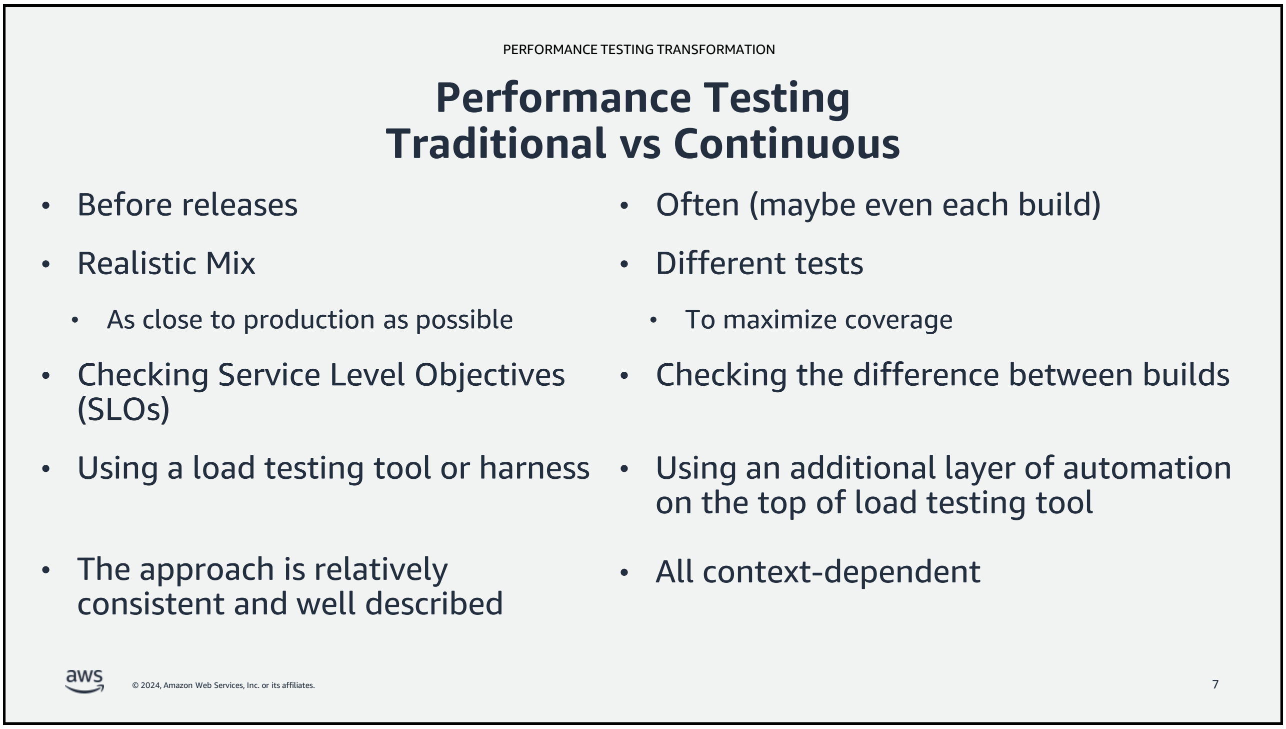
5



Continuous Performance Testing

- Continuous performance testing
  - To catch regressions early
- Collecting all info needed to investigate regressions
  - In the form convenient for further analysis
- Foundation to build further automation on the top of it
  - For further performance optimization
- All context-dependent
  - Don’t wait for an exact recipe, figure it out depending on your needs

6



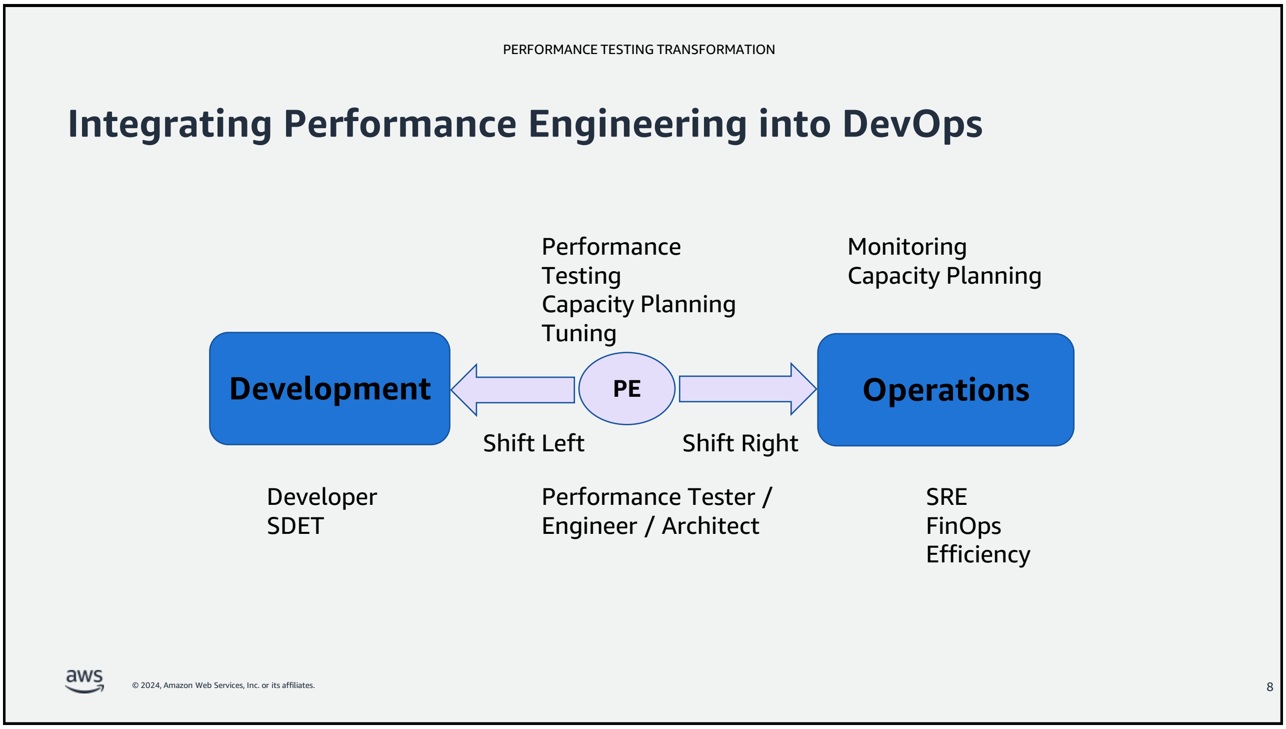
PERFORMANCE TESTING TRANSFORMATION

Performance Testing Traditional vs Continuous

- Before releases
- Realistic Mix
  - As close to production as possible
- Checking Service Level Objectives (SLOs)
- Using a load testing tool or harness
- The approach is relatively consistent and well described

- Often (maybe even each build)
- Different tests
  - To maximize coverage
- Checking the difference between builds
- Using an additional layer of automation on the top of load testing tool
- All context-dependent

7



PERFORMANCE TESTING TRANSFORMATION

Integrating Performance Engineering into DevOps

Performance Testing
Capacity Planning
Tuning

Monitoring
Capacity Planning

Development

PE

Operations

Shift Left

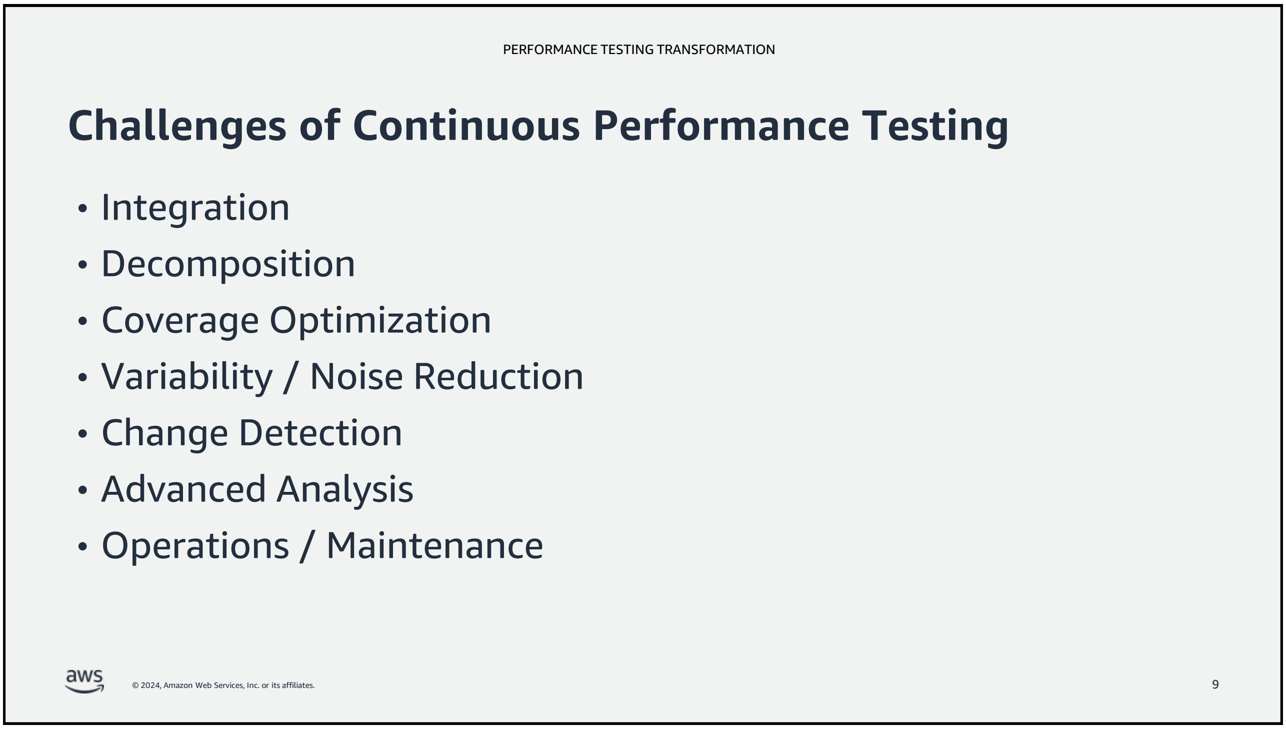
Shift Right

Developer
SDET

Performance Tester / Engineer / Architect

SRE
FinOps
Efficiency

8



PERFORMANCE TESTING TRANSFORMATION

Challenges of Continuous Performance Testing

- Integration
- Decomposition
- Coverage Optimization
- Variability / Noise Reduction
- Change Detection
- Advanced Analysis
- Operations / Maintenance

9

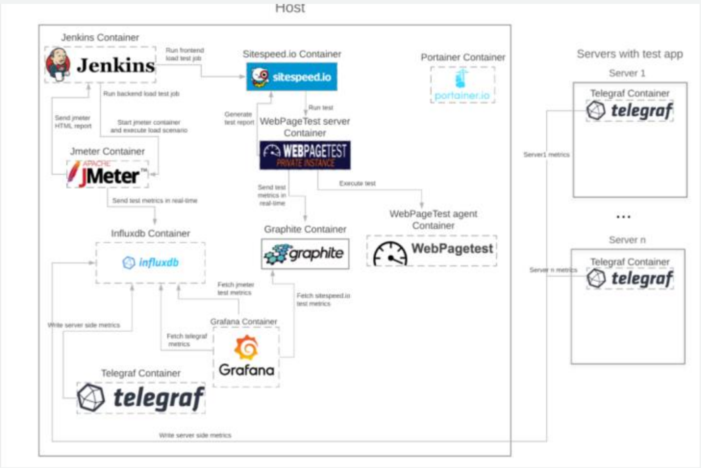
The Challenge of Integration

10

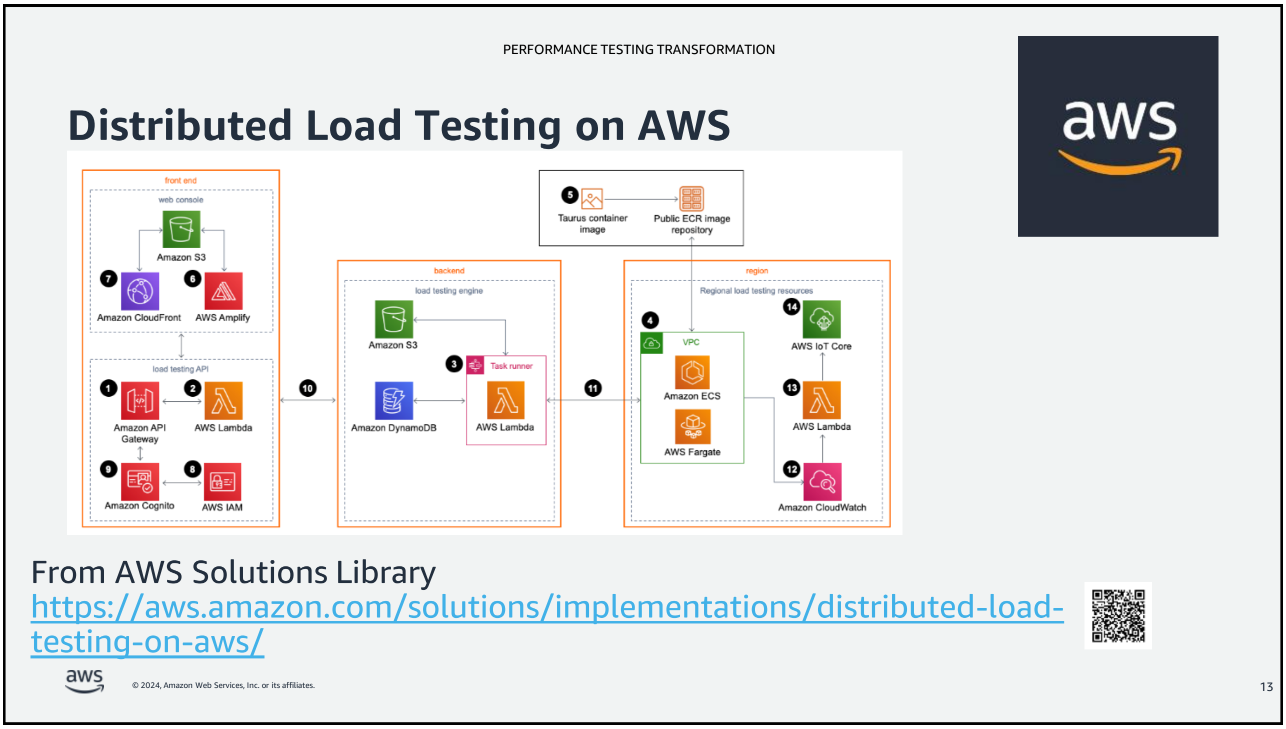
Continuous Integration: Load Testing Tools

- CI support in load testing tools
  - Integration with CI Servers (Jenkins, Hudson, etc.)
  - Automation support
- CI tools support for performance testing
  - Jenkins Performance Plugin
- Performance Testing Frameworks
  - Combining multiple tools

A Performance Testing Framework

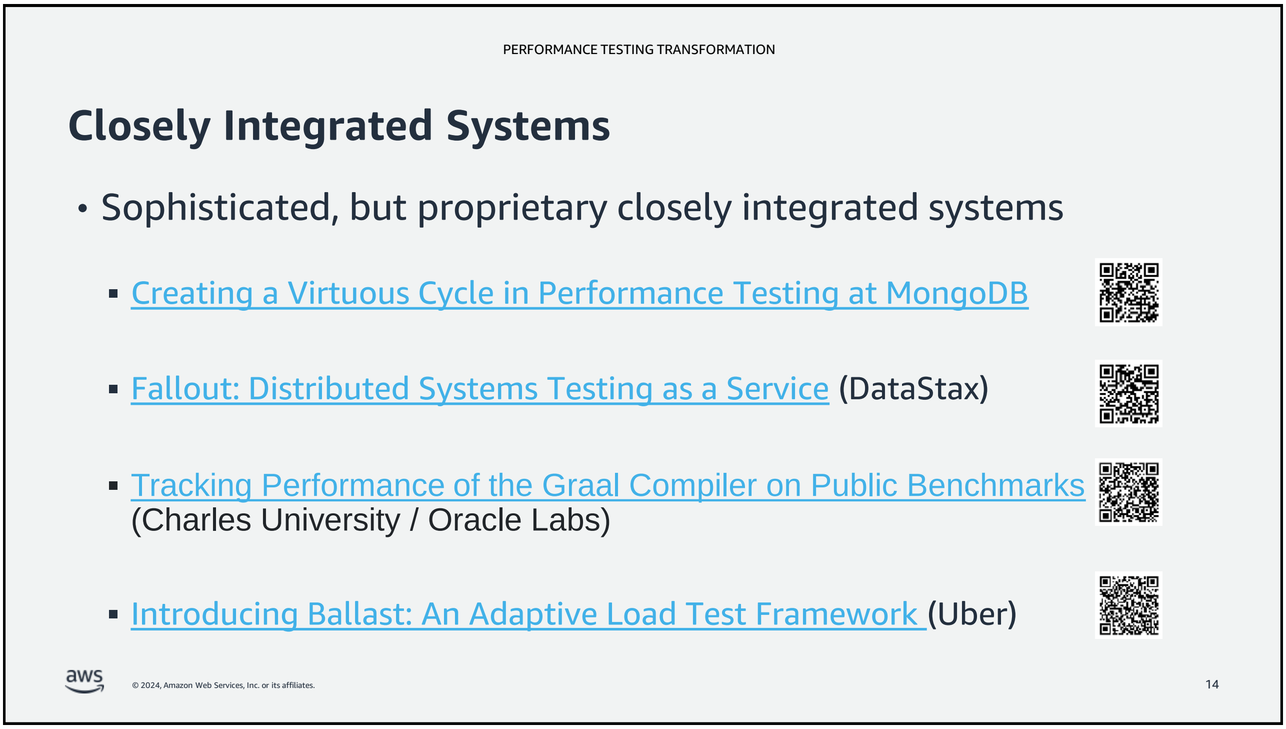


An example: https://github.com/serputko/performance-testing-framework



Distributed Load Testing on AWS

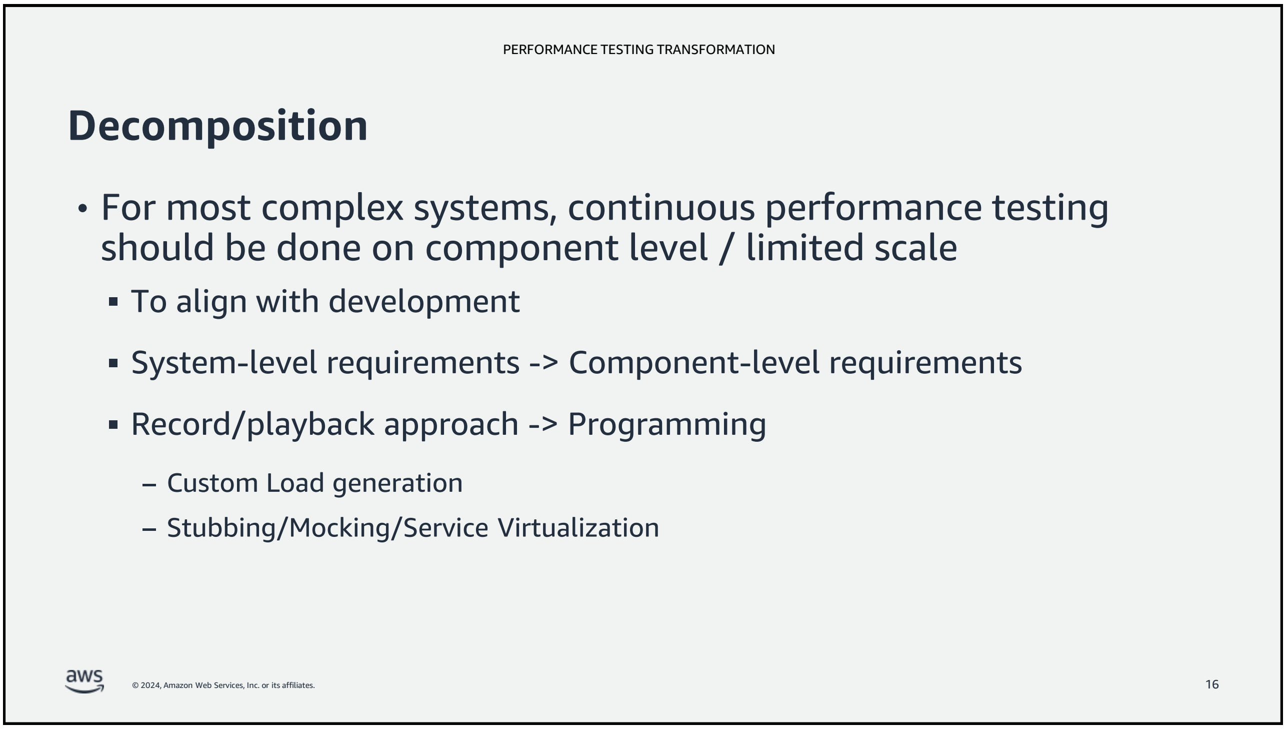
From AWS Solutions Library
https://aws.amazon.com/solutions/implementations/distributed-load-testing-on-aws/



Closely Integrated Systems

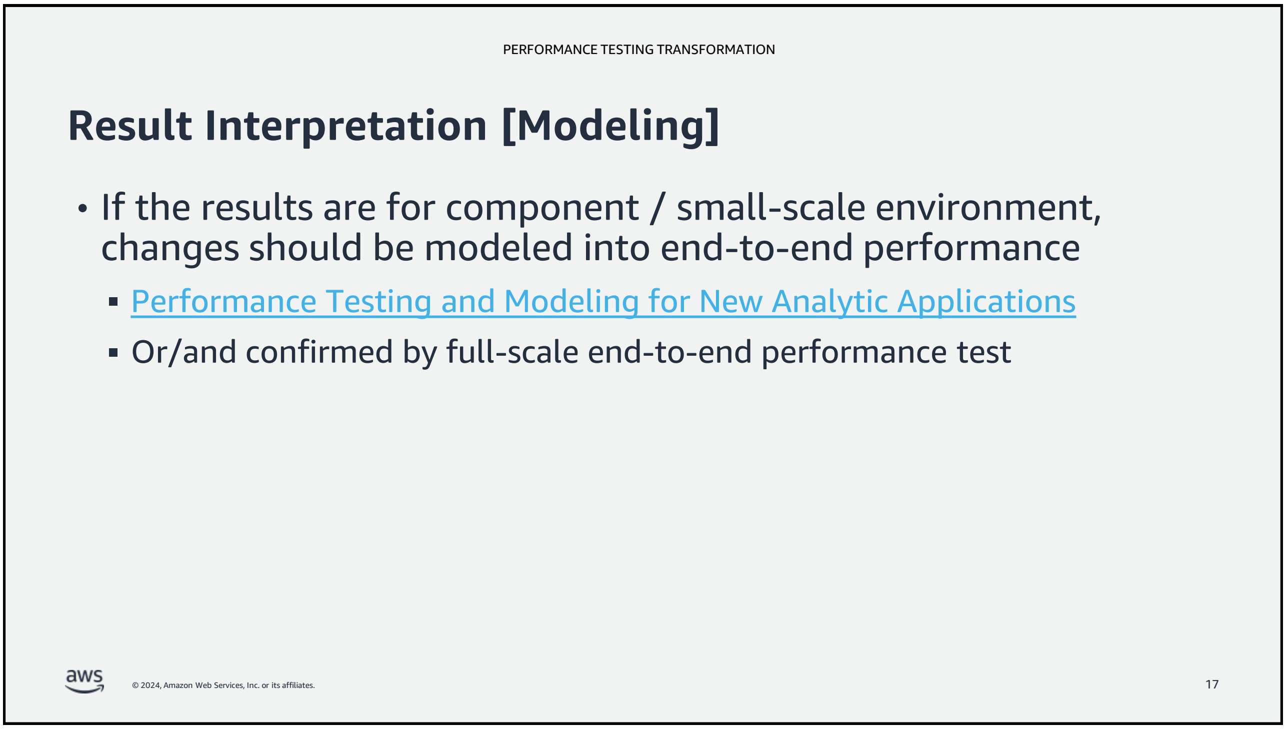
- Sophisticated, but proprietary closely integrated systems
  - Creating a Virtuous Cycle in Performance Testing at MongoDB
  - Fallout: Distributed Systems Testing as a Service (DataStax)
  - Tracking Performance of the Graal Compiler on Public Benchmarks (Charles University / Oracle Labs)
  - Introducing Ballast: An Adaptive Load Test Framework (Uber)

The Challenge of Decomposition



Decomposition

- For most complex systems, continuous performance testing should be done on component level / limited scale
  - To align with development
  - System-level requirements -> Component-level requirements
  - Record/playback approach -> Programming
    - Custom Load generation
    - Stubbing/Mocking/Service Virtualization

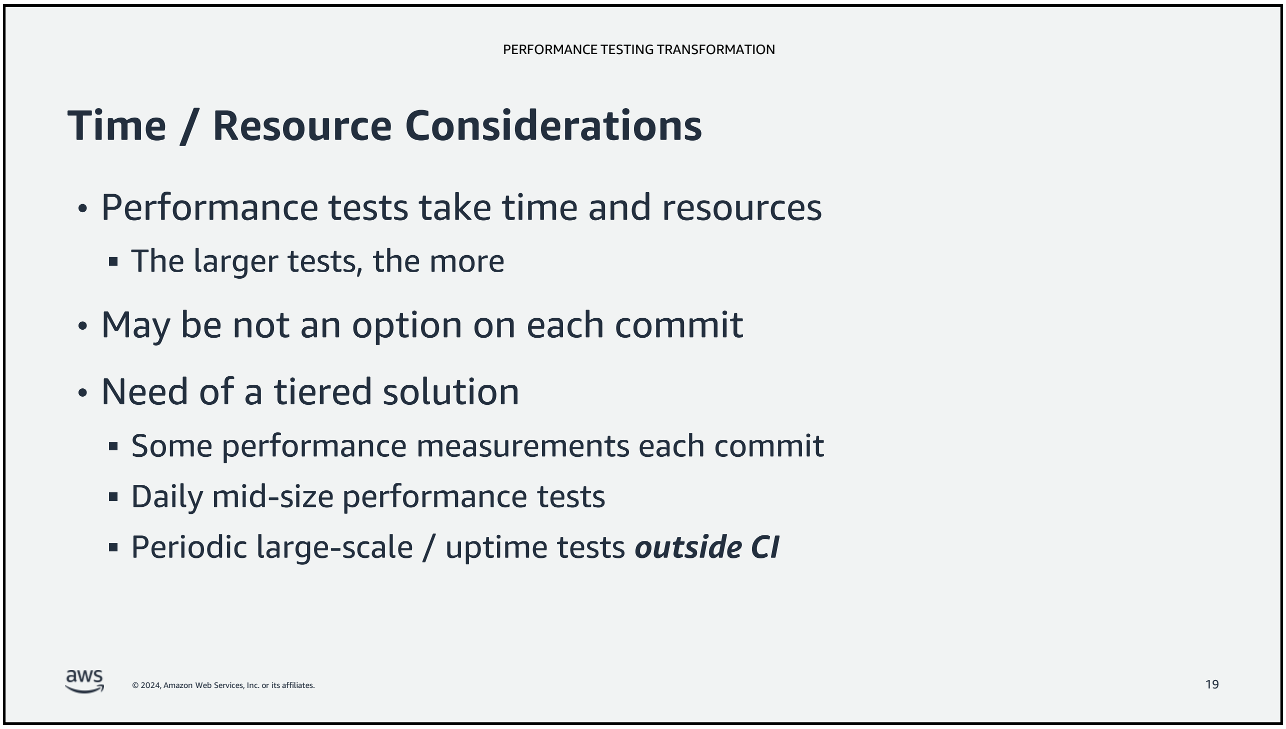


PERFORMANCE TESTING TRANSFORMATION

Result Interpretation [Modeling]

- If the results are for component / small-scale environment, changes should be modeled into end-to-end performance
  - Performance Testing and Modeling for New Analytic Applications
  - Or/and confirmed by full-scale end-to-end performance test

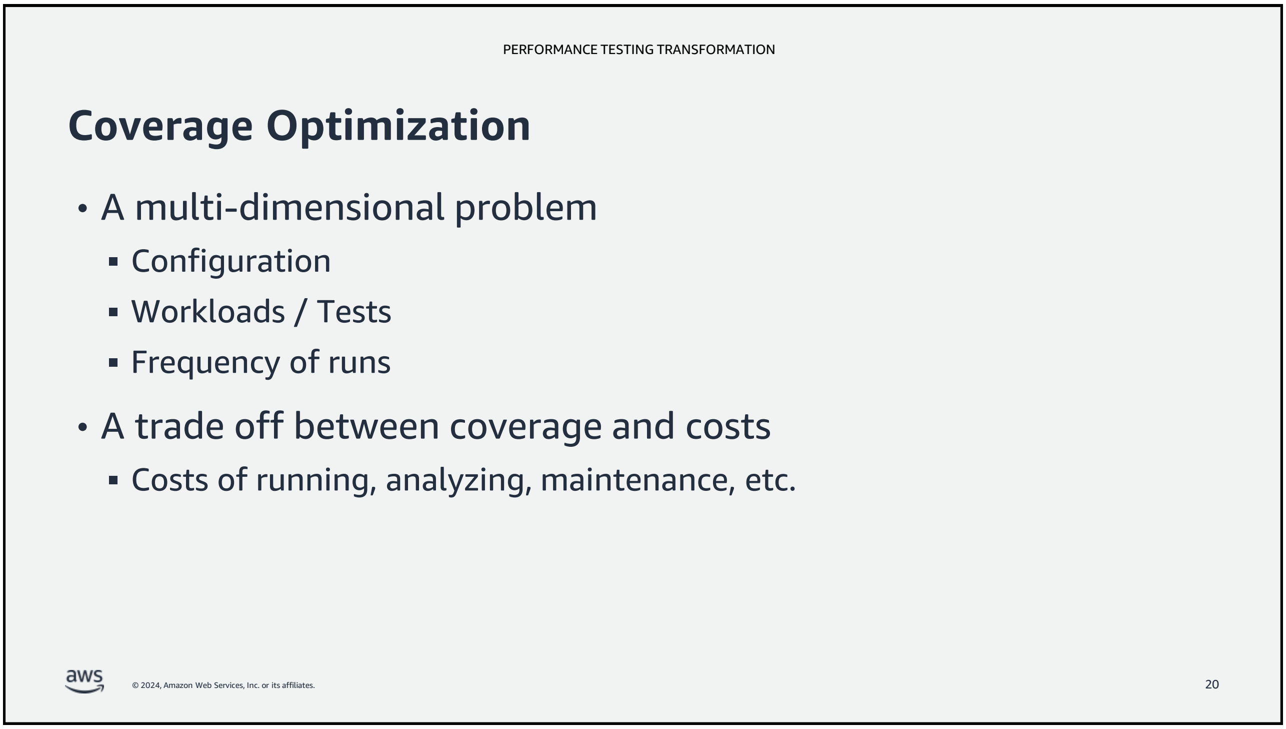
The Challenge of Coverage Optimization



Time / Resource Considerations

- Performance tests take time and resources
  - The larger tests, the more
- May be not an option on each commit
- Need of a tiered solution
  - Some performance measurements each commit
  - Daily mid-size performance tests
  - Periodic large-scale / uptime tests ***outside CI***

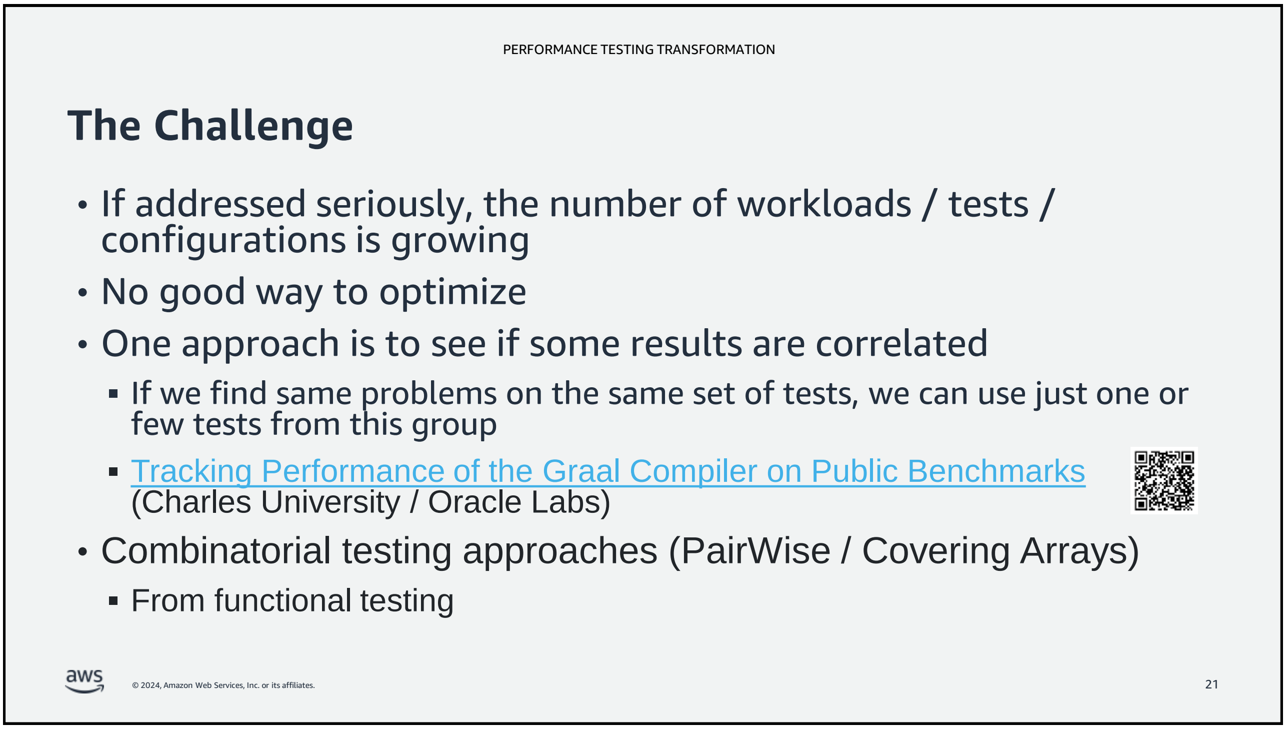
19



Coverage Optimization

- A multi-dimensional problem
  - Configuration
  - Workloads / Tests
  - Frequency of runs
- A trade off between coverage and costs
  - Costs of running, analyzing, maintenance, etc.

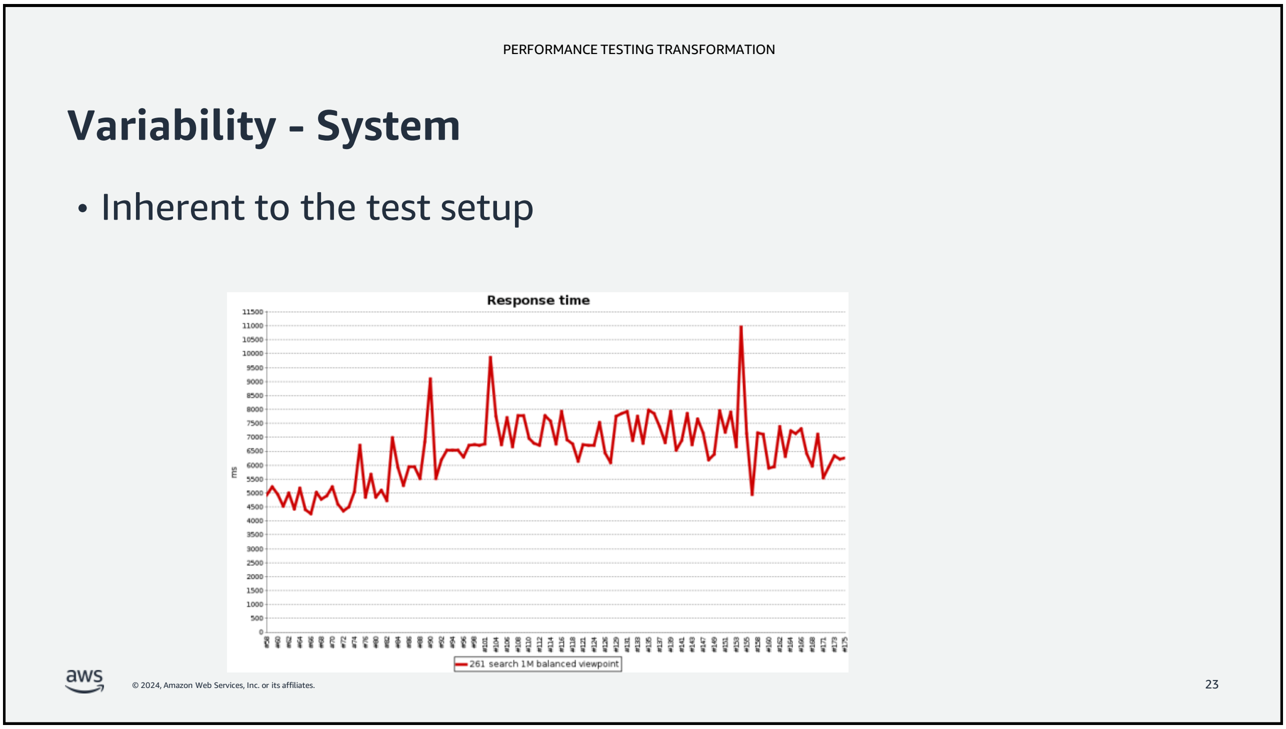
20



The Challenge

- If addressed seriously, the number of workloads / tests / configurations is growing
- No good way to optimize
- One approach is to see if some results are correlated
  - If we find same problems on the same set of tests, we can use just one or few tests from this group
  - Tracking Performance of the Graal Compiler on Public Benchmarks (Charles University / Oracle Labs)
- Combinatorial testing approaches (PairWise / Covering Arrays)
  - From functional testing

The Challenge of Variability



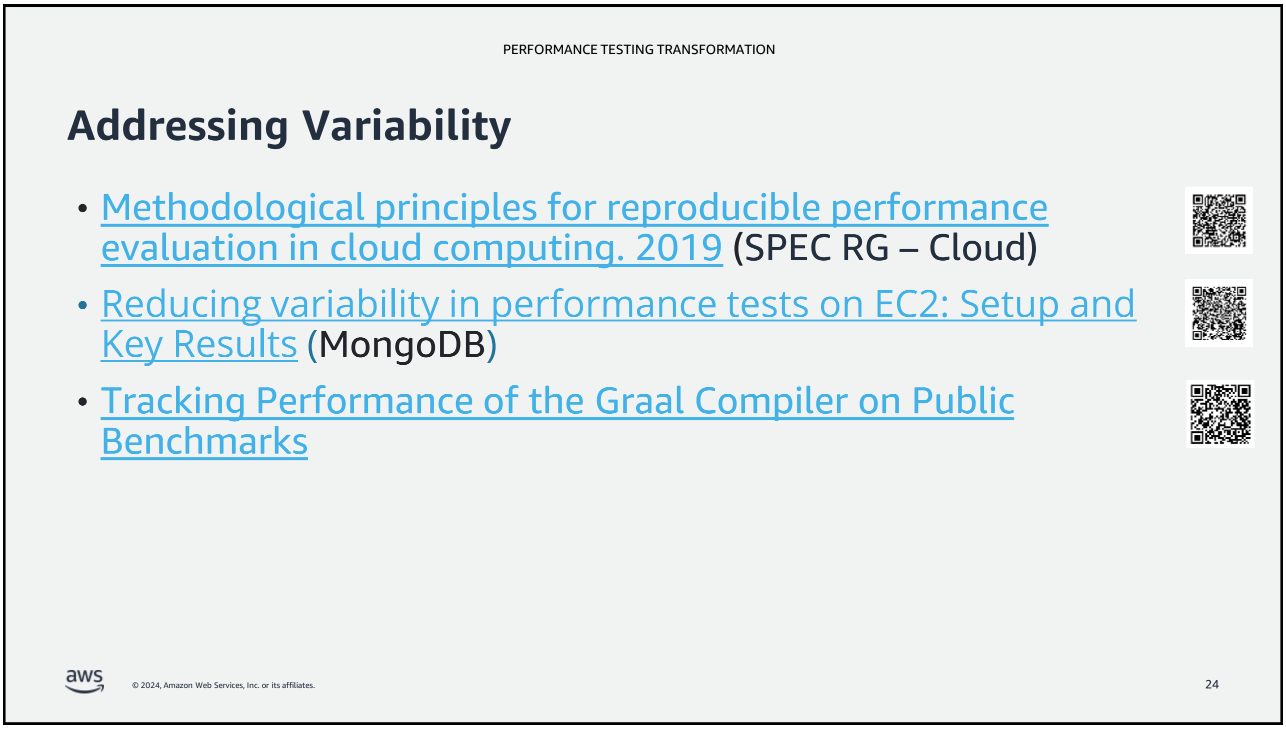
PERFORMANCE TESTING TRANSFORMATION

Variability - System

- Inherent to the test setup

23

23



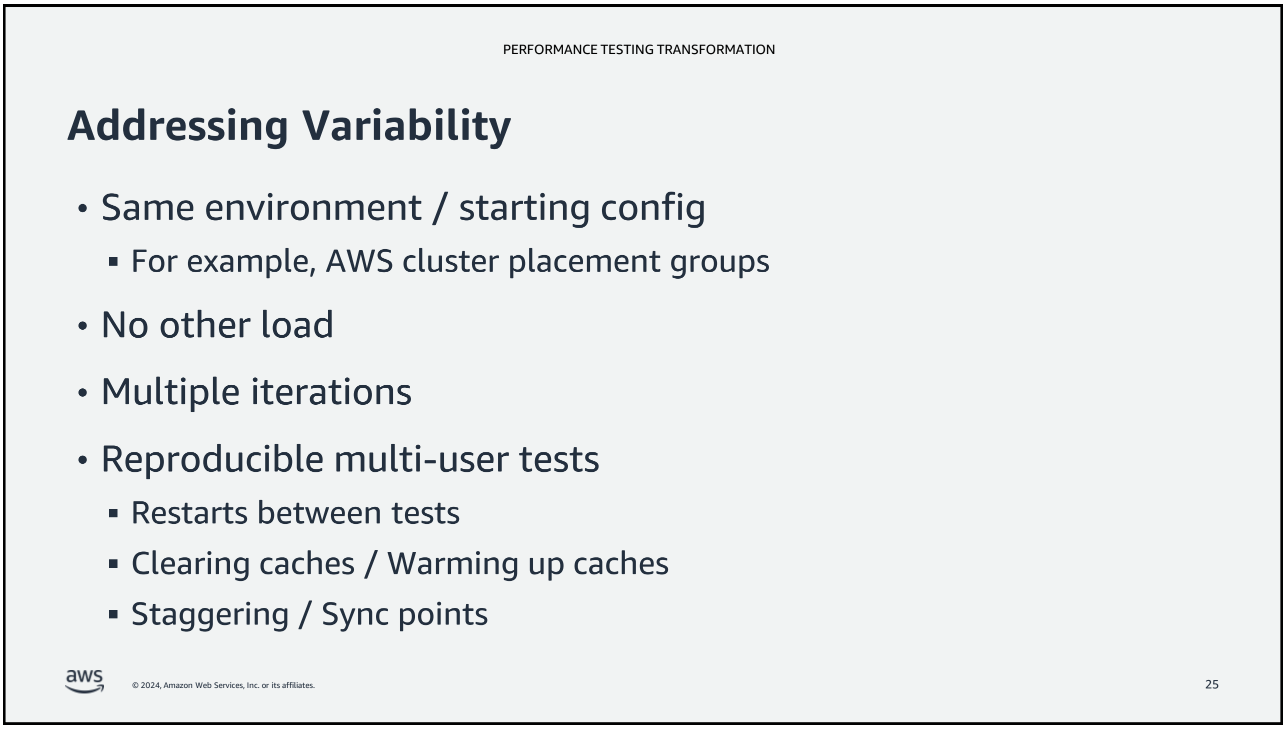
PERFORMANCE TESTING TRANSFORMATION

Addressing Variability

- Methodological principles for reproducible performance evaluation in cloud computing. 2019 (SPEC RG – Cloud)
- Reducing variability in performance tests on EC2: Setup and Key Results (MongoDB)
- Tracking Performance of the Graal Compiler on Public Benchmarks

24

24



PERFORMANCE TESTING TRANSFORMATION

Addressing Variability

- Same environment / starting config
  - For example, AWS cluster placement groups
- No other load
- Multiple iterations
- Reproducible multi-user tests
  - Restarts between tests
  - Clearing caches / Warming up caches
  - Staggering / Sync points

25

The Challenge of Change Detection

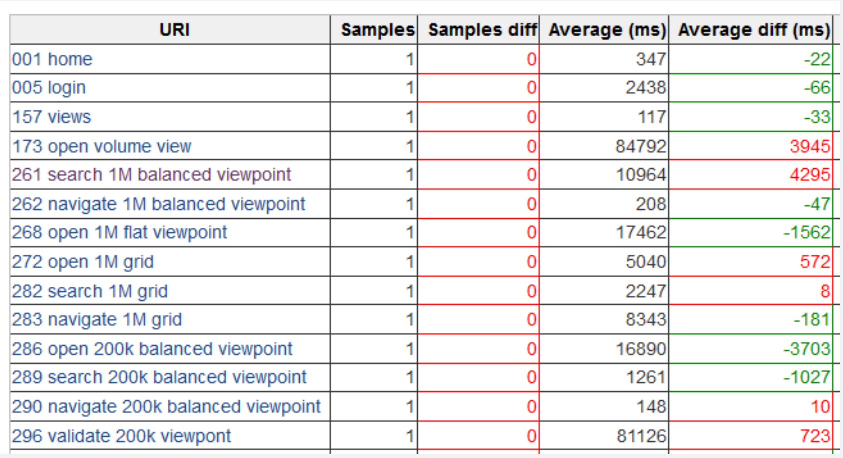
26

Complex Results

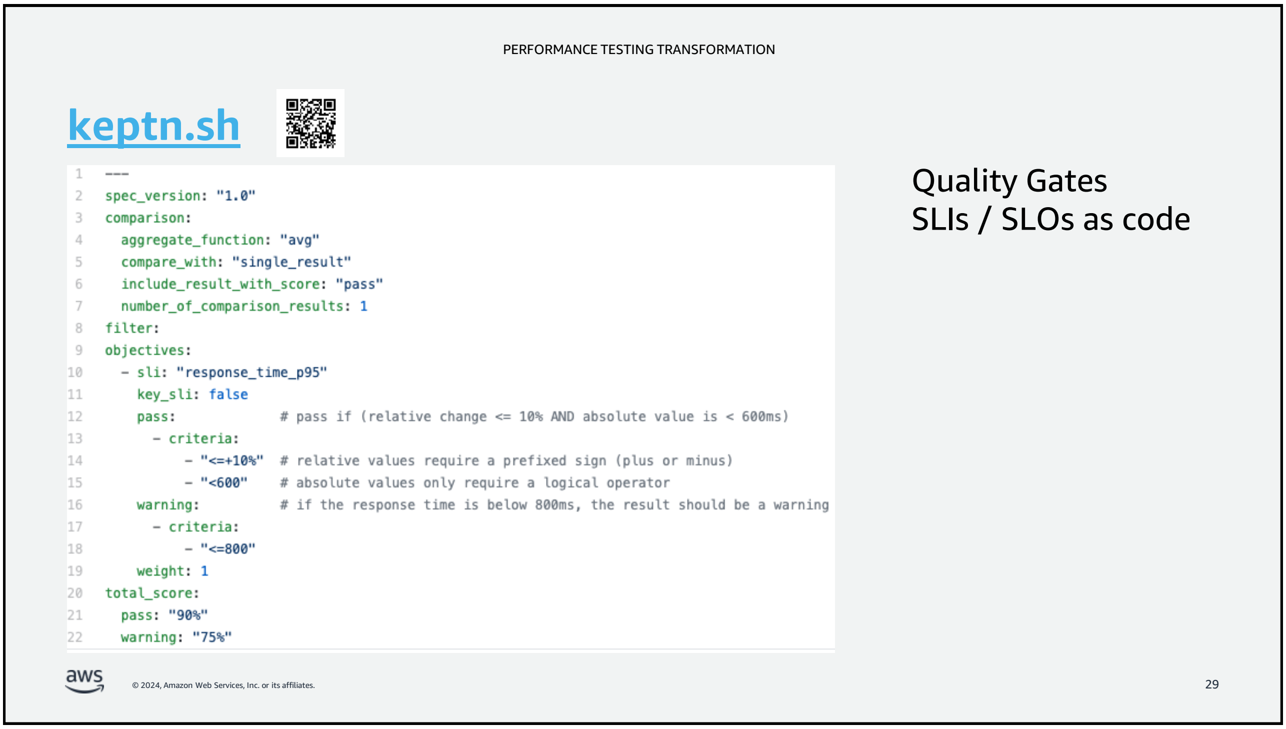
- No easy pass/fail
  - Individual responses, monitoring results, errors, etc.
- No easy comparison
  - Against SLA
  - Between builds
- Variability

Simple Comparison

Jenkins Performance Plugin



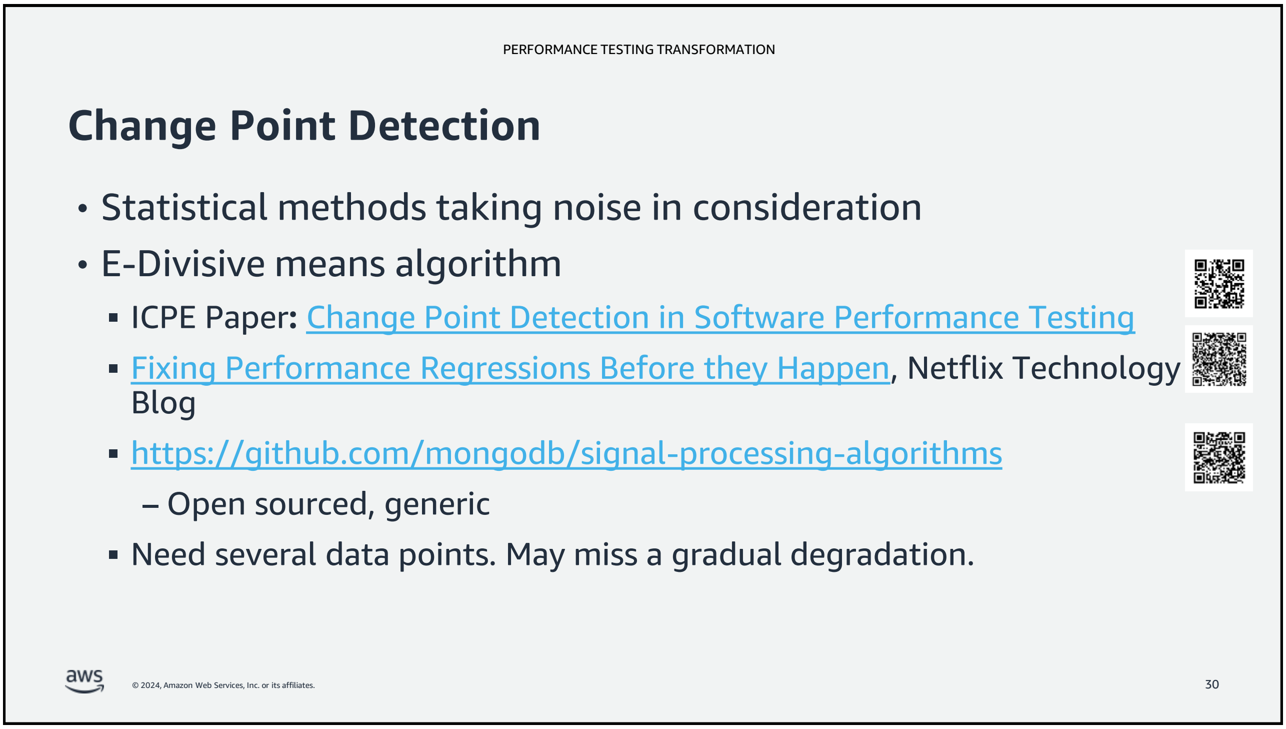
URI	Samples	Samples diff	Average (ms)	Average diff (ms)
001 home	1	0	347	-22
005 login	1	0	2438	-66
157 views	1	0	117	-33
173 open volume view	1	0	84792	3945
261 search 1M balanced viewpoint	1	0	10964	4295
262 navigate 1M balanced viewpoint	1	0	208	-47
268 open 1M flat viewpoint	1	0	17462	-1562
272 open 1M grid	1	0	5040	572
282 search 1M grid	1	0	2247	8
283 navigate 1M grid	1	0	8343	-181
286 open 200k balanced viewpoint	1	0	16890	-3703
289 search 200k balanced viewpoint	1	0	1261	-1027
290 navigate 200k balanced viewpoint	1	0	148	10
296 validate 200k viewpont	1	0	81126	723



keptn.sh

```
---
spec_version: "1.0"
comparison:
  aggregate_function: "avg"
  compare_with: "single_result"
  include_result_with_score: "pass"
  number_of_comparison_results: 1
filter:
objectives:
  - sli: "response_time_p95"
    key_sli: false
    pass:             # pass if (relative change <= 10% AND absolute value is < 600ms)
      - criteria:
          - "<=+10%"  # relative values require a prefixed sign (plus or minus)
          - "<600"    # absolute values only require a logical operator
    warning:          # if the response time is below 800ms, the result should be a warning
      - criteria:
          - "<=800"
    weight: 1
total_score:
  pass: "90%"
  warning: "75%"
```

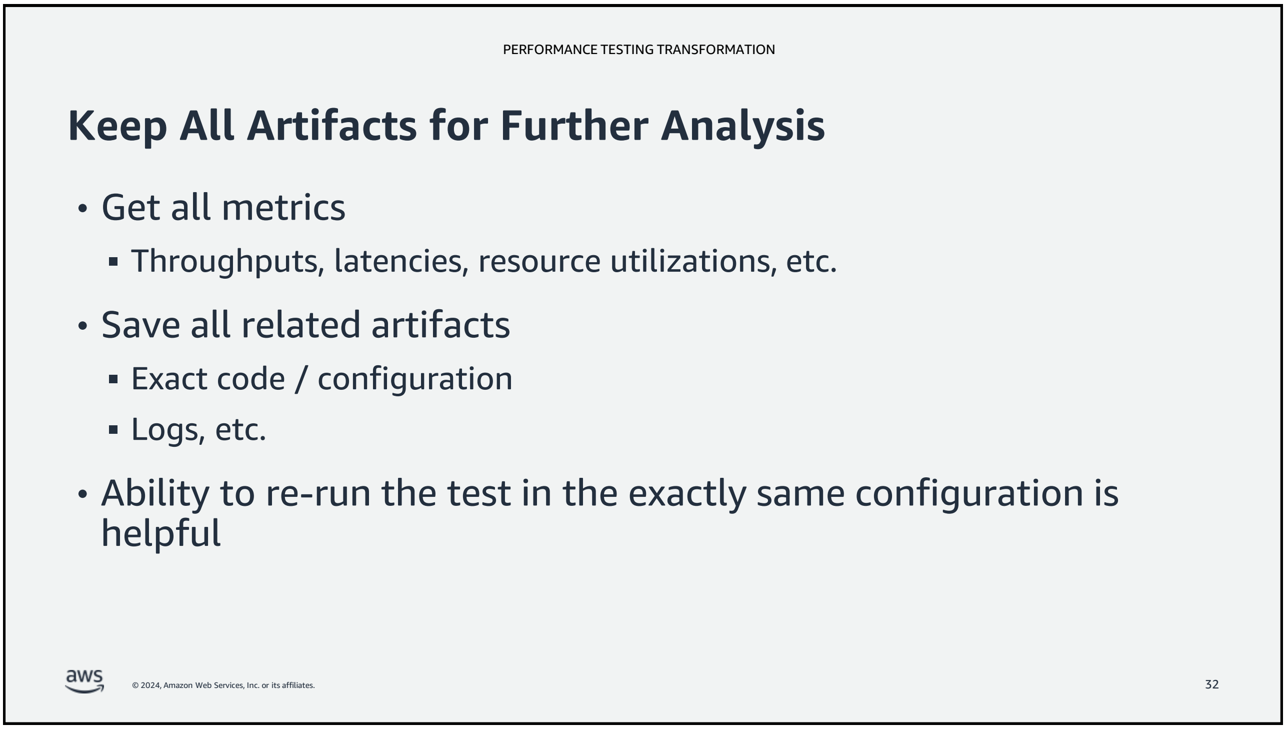
Quality Gates
SLIs / SLOs as code



Change Point Detection

- Statistical methods taking noise in consideration
- E-Divisive means algorithm
  - ICPE Paper: Change Point Detection in Software Performance Testing
  - Fixing Performance Regressions Before they Happen, Netflix Technology Blog
  - https://github.com/mongodb/signal-processing-algorithms
    - Open sourced, generic
  - Need several data points. May miss a gradual degradation.

The Challenge of Advanced Analysis



Keep All Artifacts for Further Analysis

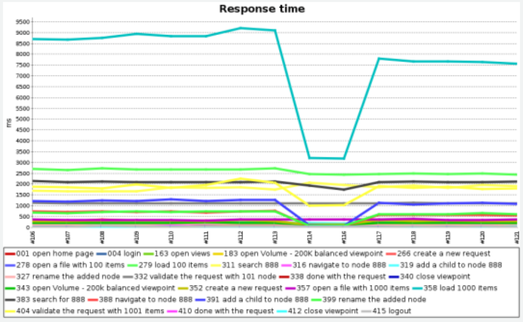
- Get all metrics
  - Throughputs, latencies, resource utilizations, etc.
- Save all related artifacts
  - Exact code / configuration
  - Logs, etc.
- Ability to re-run the test in the exactly same configuration is helpful

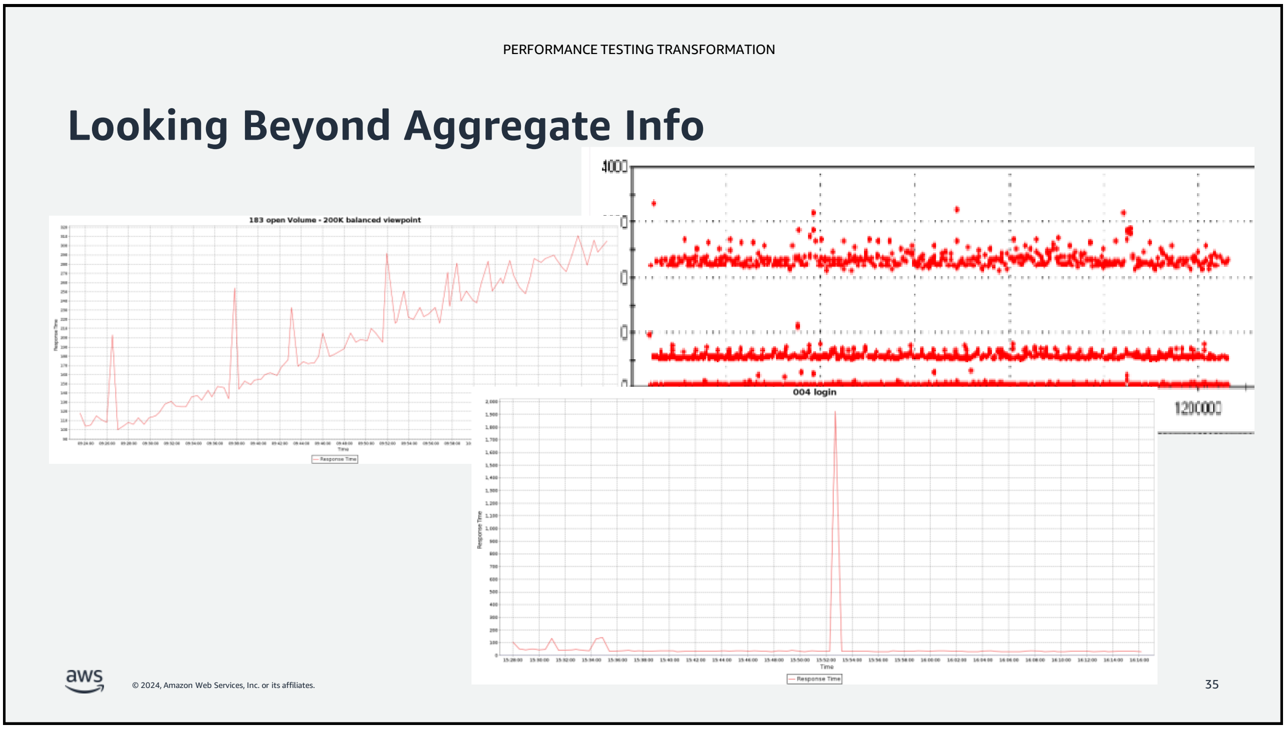
Root Cause Analysis

- Collecting artifacts to do root cause analysis
- Insights snapshots
  - Flamegraphs (perf, eBPF)
- Continuous Profiling
  - Java Flight Recorder
  - APM
  - Tracing
  - Observability
  - eBPF-based tools

Visualization

- Visualizing systems and software performance - Report on the GI-Dagstuhl
- Sometimes helps to catch an issue





PERFORMANCE TESTING TRANSFORMATION

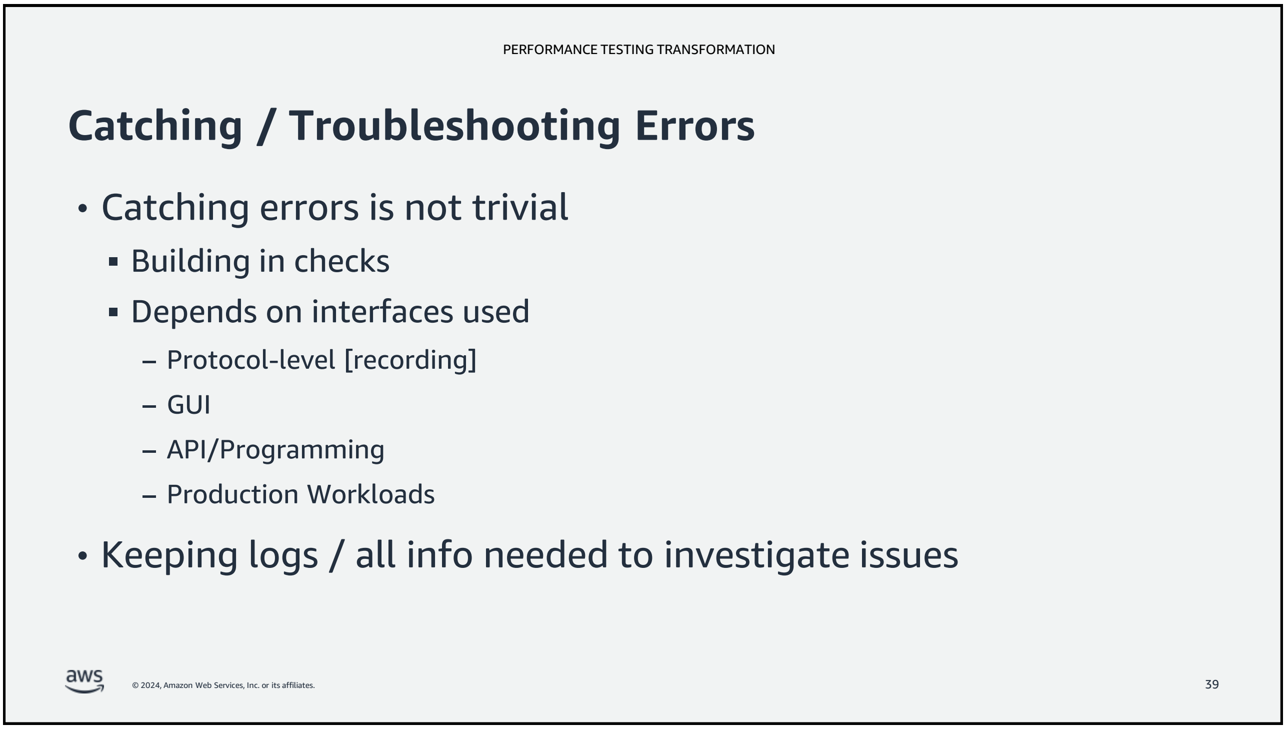
Looking Beyond Aggregate Info

The Challenge of Operations and Maintenance

Operations

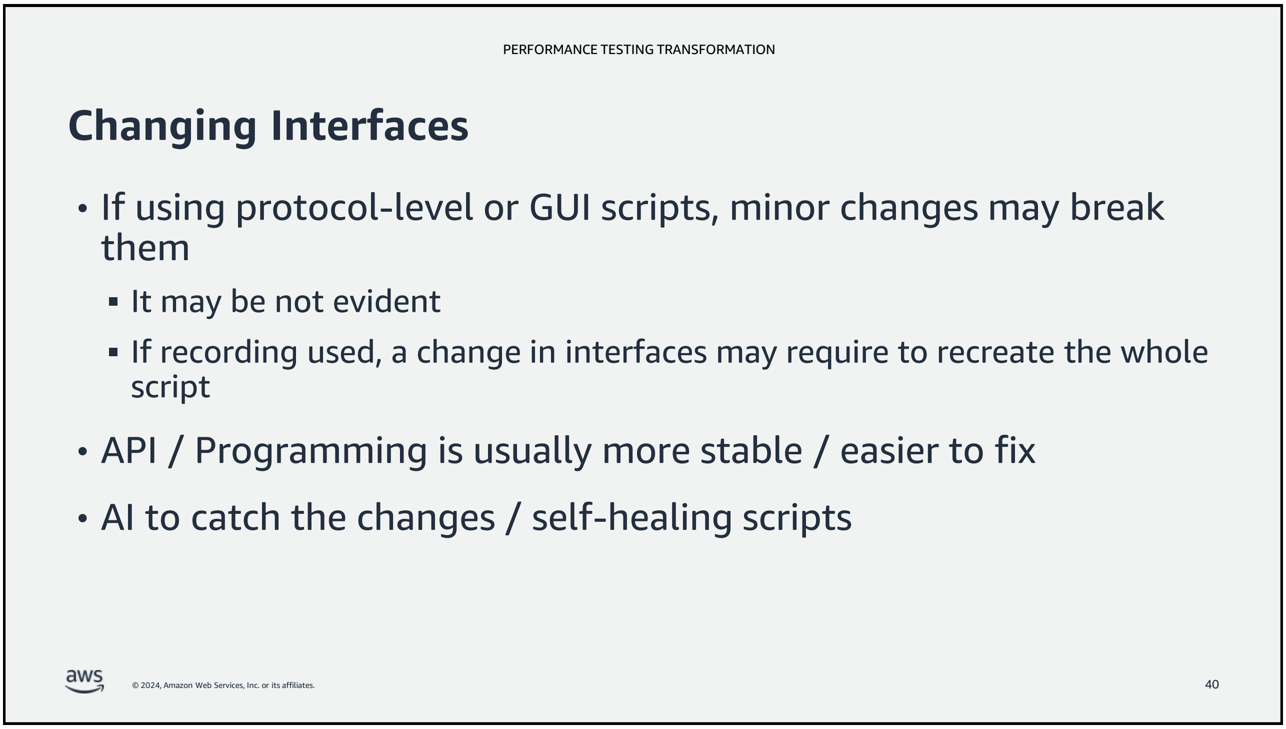
- Scheduling / execution tests
- Results analysis
- Triaging and escalating issues
- Maintenance

Coverage / Maintenance Trade-Off



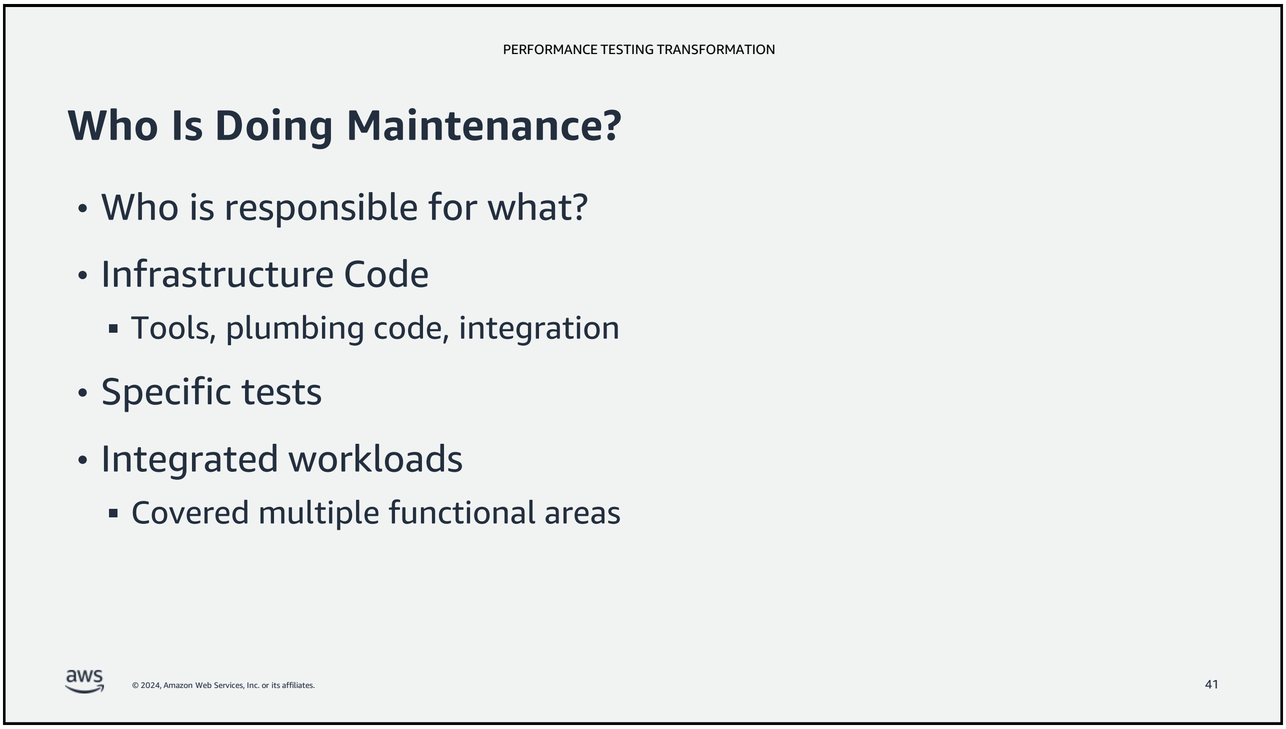
Catching / Troubleshooting Errors

- Catching errors is not trivial
  - Building in checks
  - Depends on interfaces used
    - Protocol-level [recording]
    - GUI
    - API/Programming
    - Production Workloads
- Keeping logs / all info needed to investigate issues



Changing Interfaces

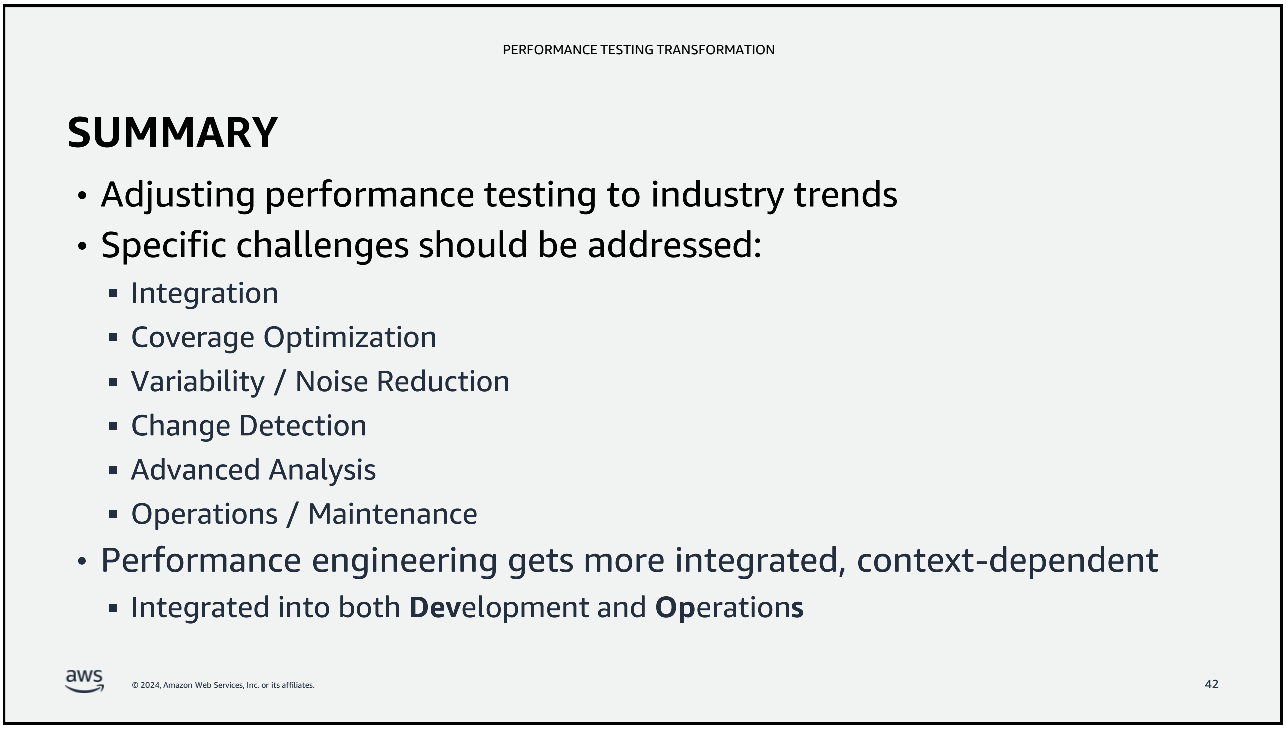
- If using protocol-level or GUI scripts, minor changes may break them
  - It may be not evident
  - If recording used, a change in interfaces may require to recreate the whole script
- API / Programming is usually more stable / easier to fix
- AI to catch the changes / self-healing scripts



PERFORMANCE TESTING TRANSFORMATION

Who Is Doing Maintenance?

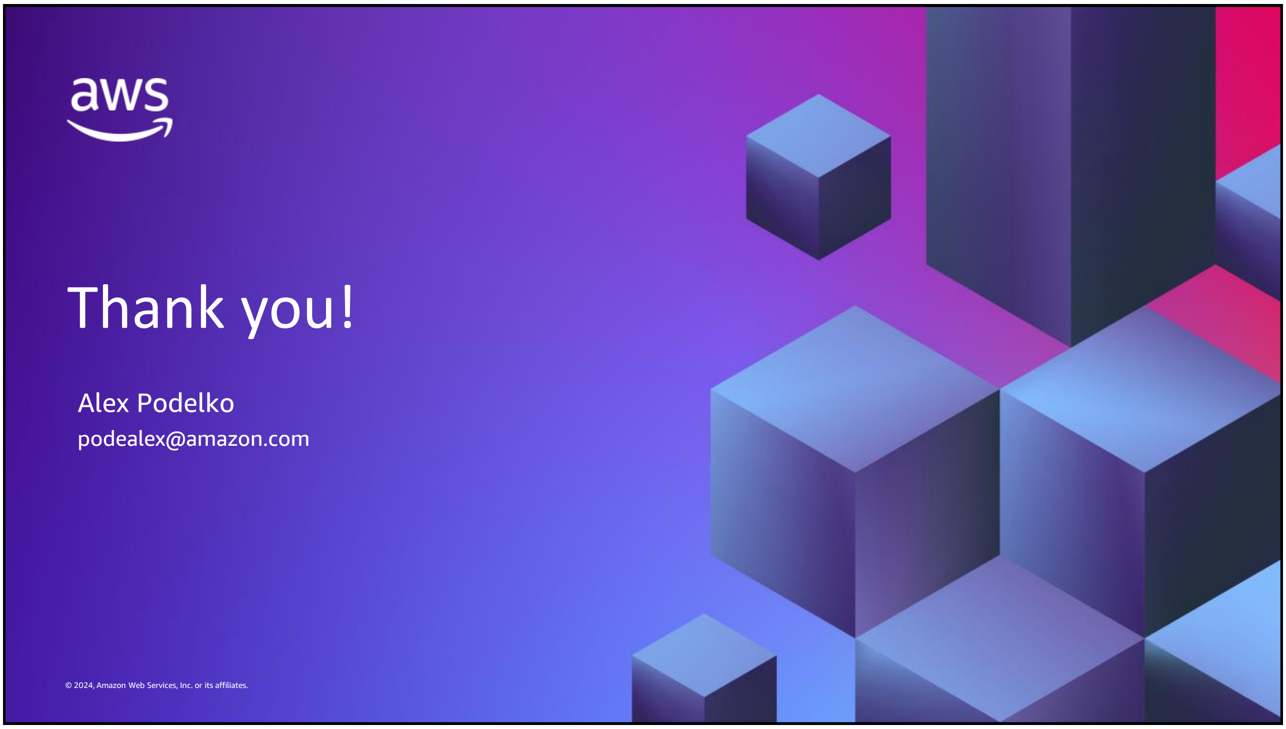
- Who is responsible for what?
- Infrastructure Code
  - Tools, plumbing code, integration
- Specific tests
- Integrated workloads
  - Covered multiple functional areas



PERFORMANCE TESTING TRANSFORMATION

SUMMARY

- Adjusting performance testing to industry trends
- Specific challenges should be addressed:
  - Integration
  - Coverage Optimization
  - Variability / Noise Reduction
  - Change Detection
  - Advanced Analysis
  - Operations / Maintenance
- Performance engineering gets more integrated, context-dependent
  - Integrated into both **Dev**elopment and **Op**eration**s**



Thank you!

Alex Podelko
podealex@amazon.com