How we streamlined our SDLC with Observability

from GitHub via Jenkins, Harbor, Argo to K8S

PRESENTER

Michael Gläss

Chief Product Architect

PRESENTER

Andreas Grabner

CNCF-Ambassador and DevRel

Agenda

- Intro
  - About us
  - Problem Statement
  - Architecture (SDLC, Tooling Landscape, Staging)
  - Target
- Where we started
- Individual Tool Monitoring & Observability
  - Optimization potential
- Deployment/Runtime view
  - Optimization potential
- Artifact flow detection
  - SDLC Events
  - Optimization potential
  - Quality Gates
- Our learnings
- Q&A

Intro

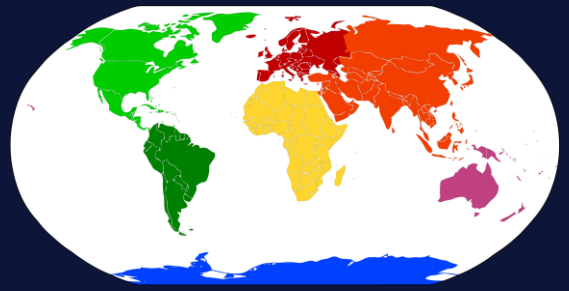
Facts about Dynatrace – SaaS only

2000
Developers

300
Builds / Day

250+
Services / Stage

Cloud Vendors

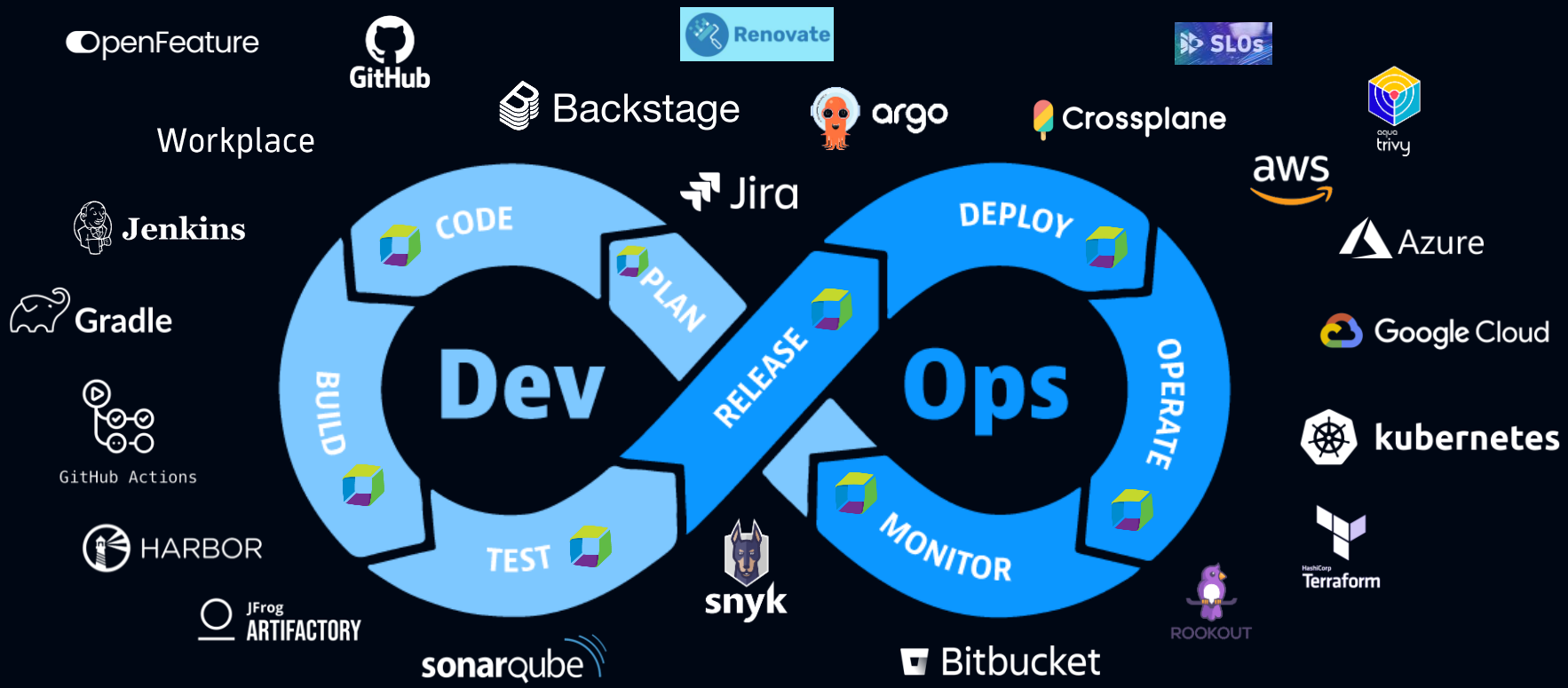

On every continent

7+2+2 dev,
3x3 hardening,
3 bugfix, X prod
Stages

Tech Stack: Cloud Native mixed with Managed Cloud Services

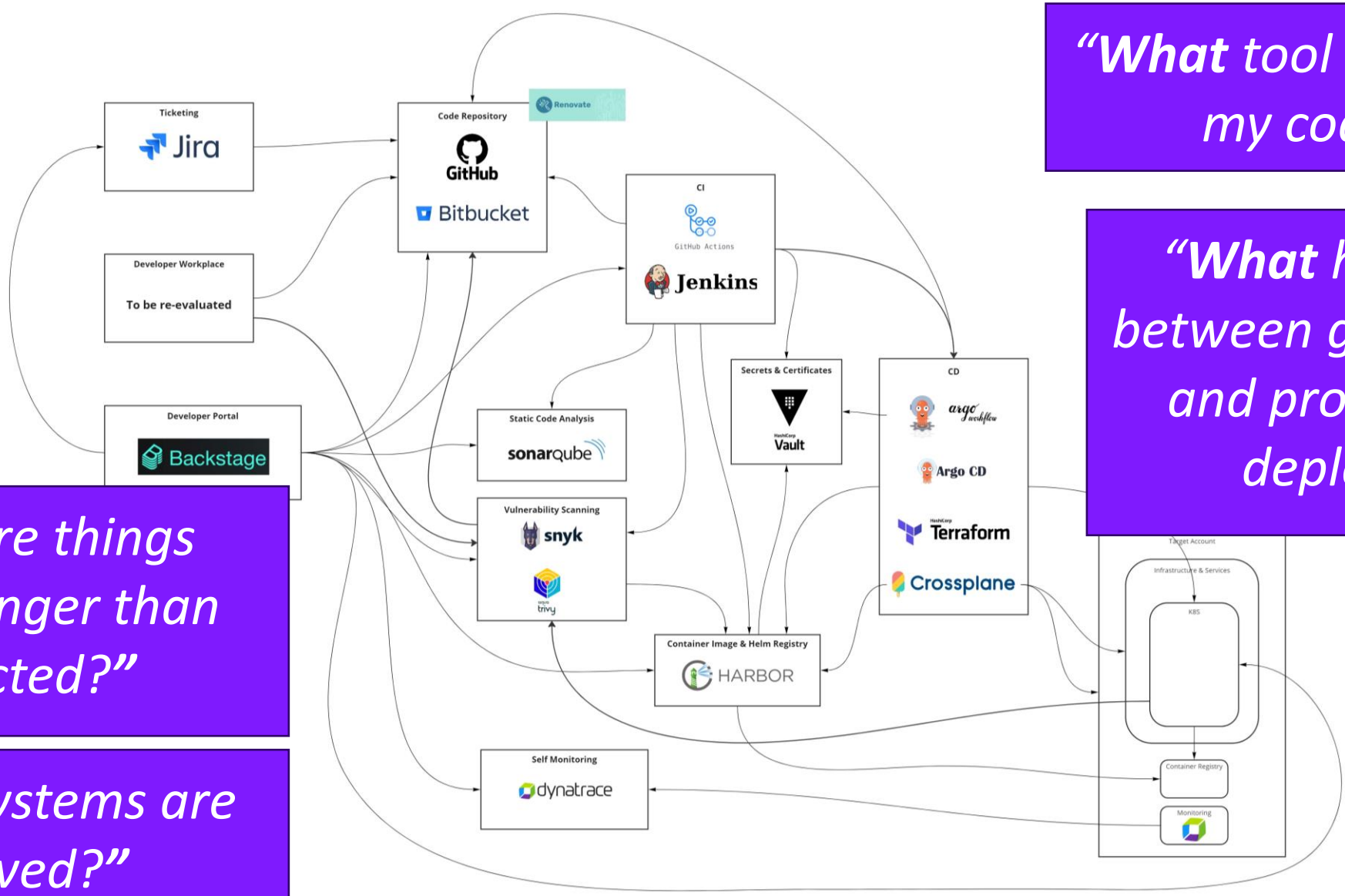
Every 2 weeks
Prod-Rollouts

DYNATRACE – SDLC (SOFTWARE DEVELOPMENT LIFECYCLE)



dynatrace

Challenges in our complex “Tooling Landscape”



*“**What** tool processed my code?”*

*“**What** happens between git commit and production deploy?”*

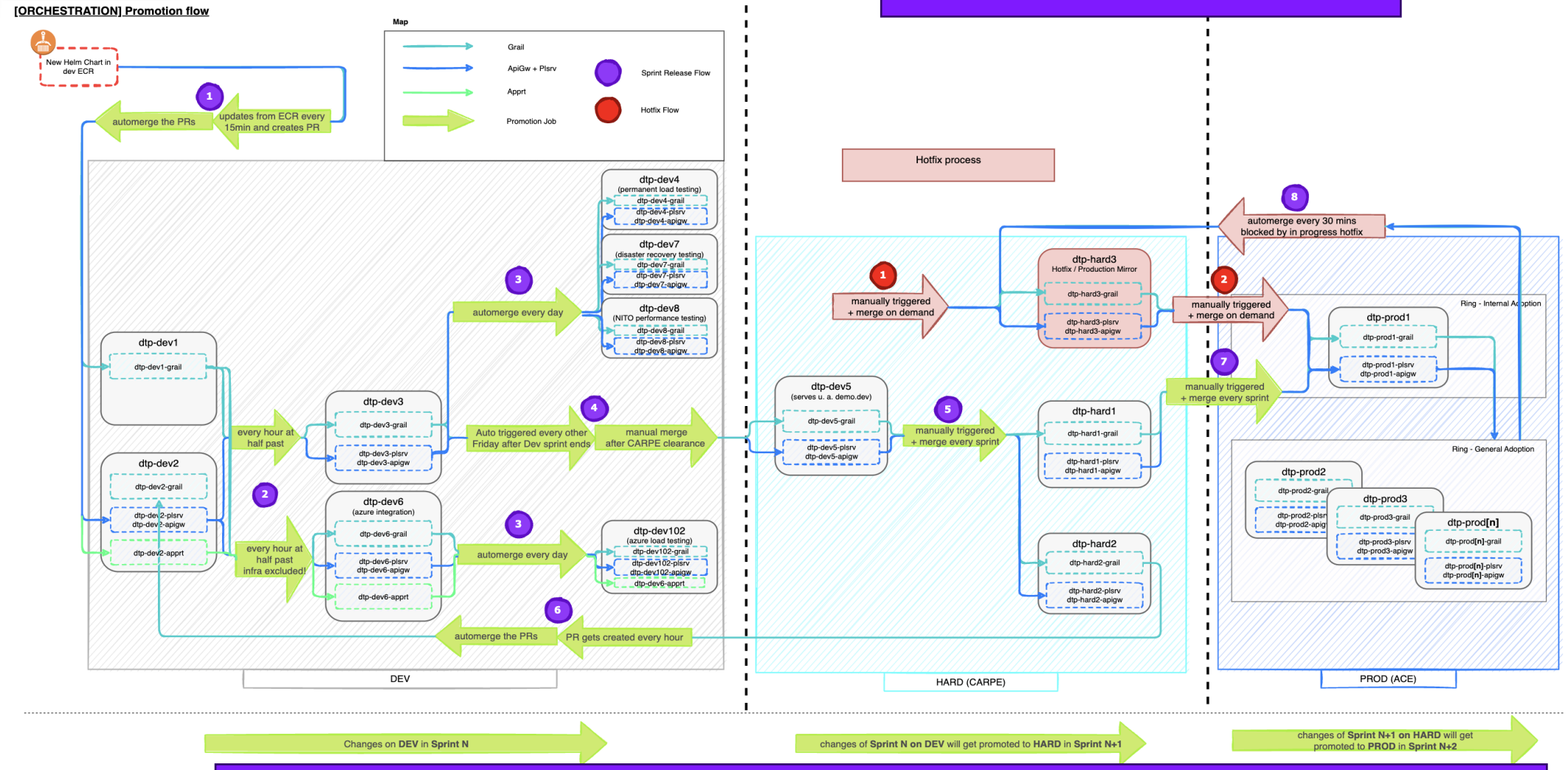
*“**Why** are things taking longer than expected?”*

*“**Which** systems are involved?”*

*Depicts only a subset of the overarching staging process.

Challenges in staging and orchestration

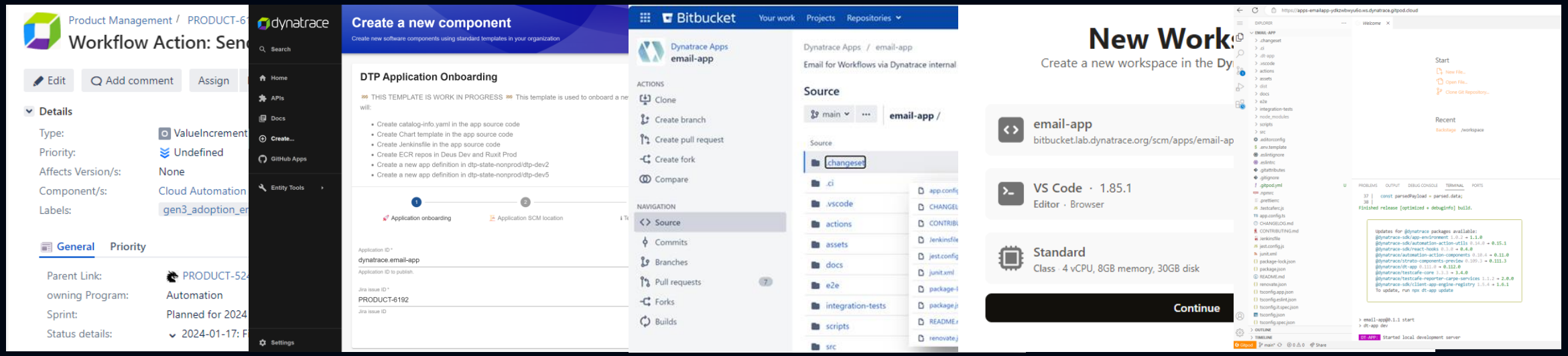
“Where is my app?”



“Why is my service still in development and not in test?”

JUNO- Developer Platform @ Dynatrace

Jira Backstage GitHub Bitbucket Workplace



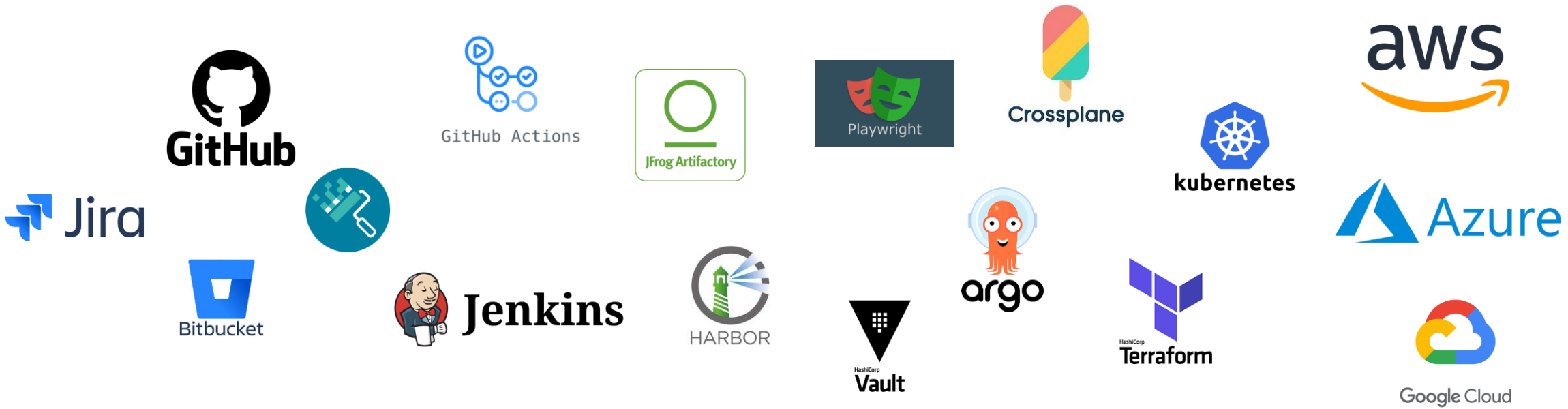
Jenkins Gradle sonarqube JFrog ARTIFACTORY GitHub Actions HARBOR Argo CD Kyverno Crossplane Open Feature aws Azure Google Cloud

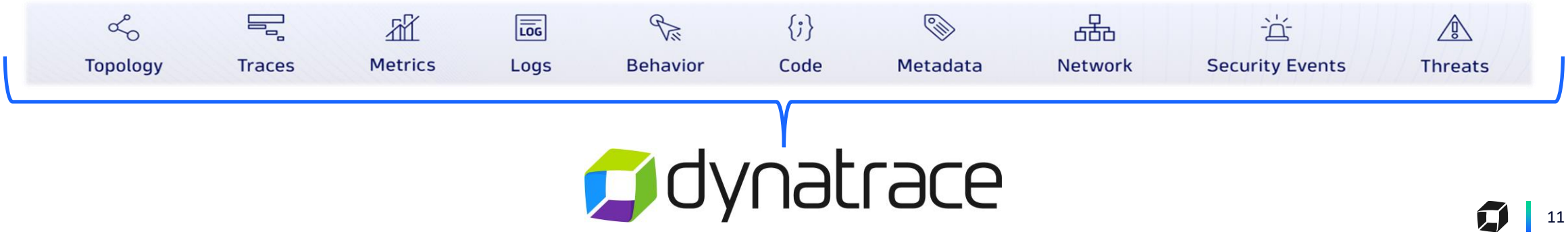
Where we started

Individual tool monitoring & observability

Tooling & System observability

*"**Every tool** and system that is involved in the DevOps cycle **needs to get observed** and the **data collected** towards a **single source of truth**."*





Tip: Platform Observability from git to prod

git commit

Backstage

Jenkins

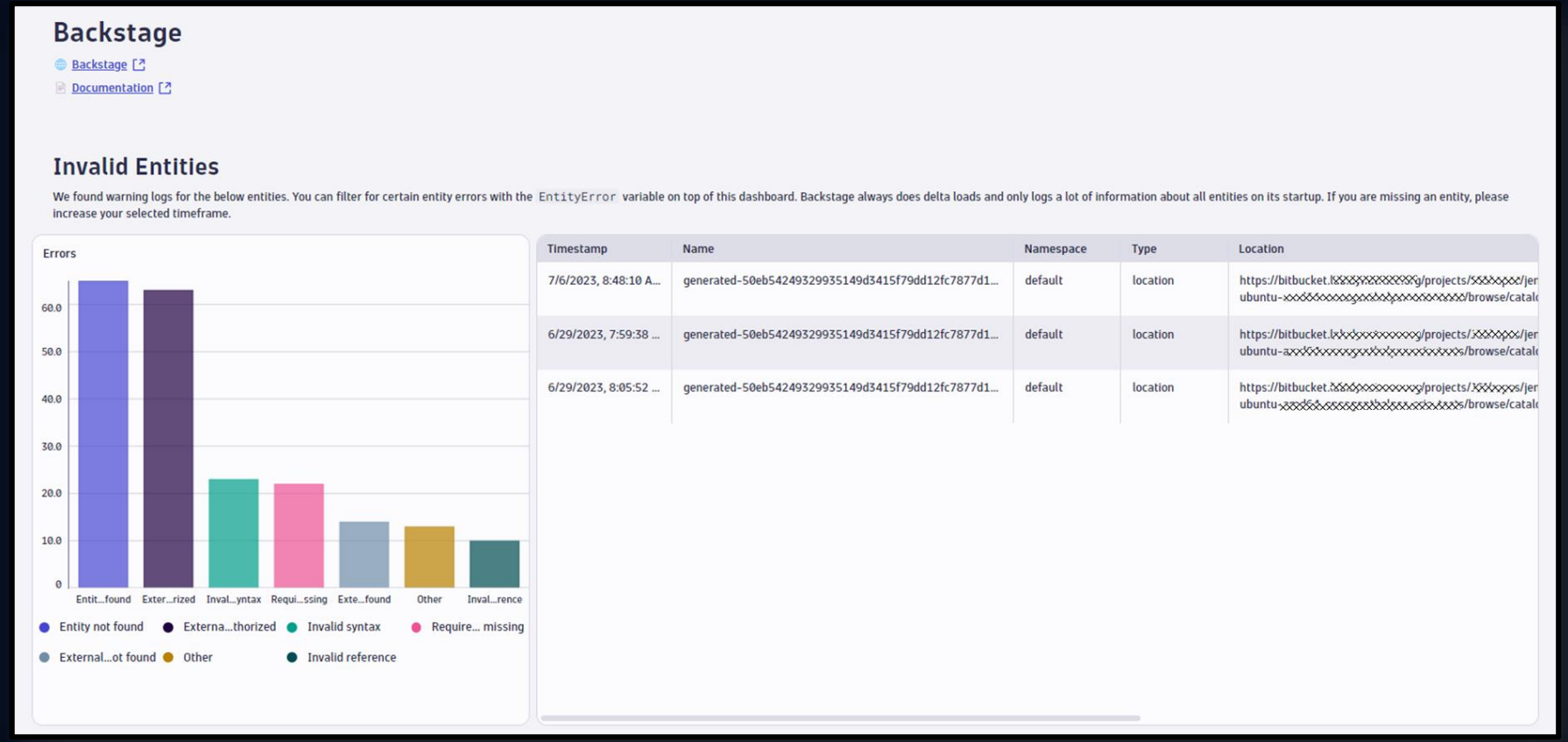
HARBOR

Kyverno

Argo CD

Crossplane

Platform Engineering: Are all templates properly configured and used?



Tip: Platform Observability from git to prod

git commit

Backstage

Jenkins

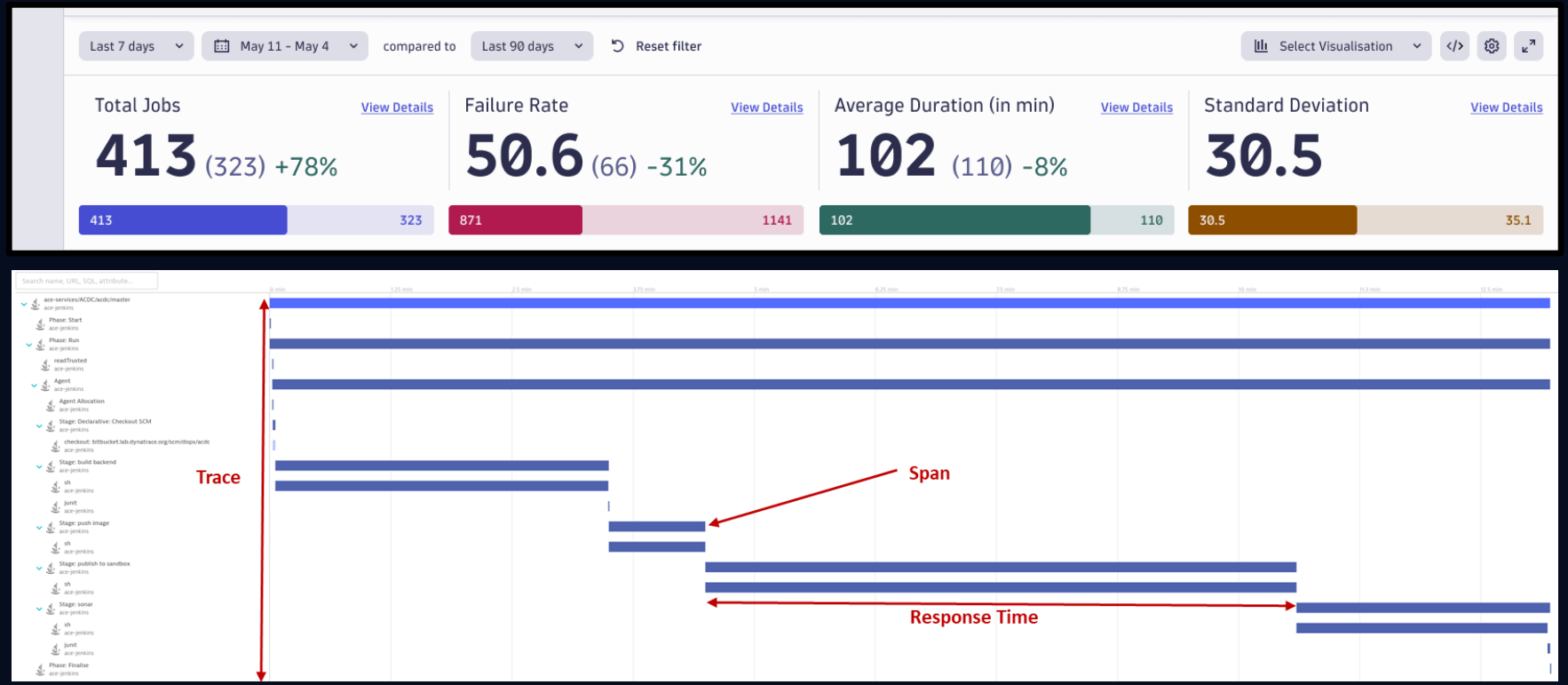
HARBOR

Kyverno

Argo CD

Crossplane

Platform Engineering: Do we have any issues and bottlenecks in our pipelines?



Tip: Platform Observability from git to prod

git commit

Backstage

Jenkins

HARBOR

Kyverno

Argo CD

Crossplane

Platform Engineering: Is our container registry running smoothly?

timestamp	content	dt.entity.cloud_application	dt.entity.cloud_application_instance
1/18/2024, 6:48:58 PM	10.44.72.254 - - [18/Jan/2024:17:48:58 +0000] "GET / HTTP/1.1" 200 785 "-" "kube-probe/1.27+"	CLOUD_APPLICATION-CCE4F8371F30A817	CLOUD_APPLICATION_INSTANCE-A33ADC0C699062BF
1/18/2024, 6:48:59 PM	100.127.66.20 - - [18/Jan/2024:17:48:59 +0000] "GET / HTTP/1.1" 200 657 "-" "Go-http-client/1.1"	CLOUD_APPLICATION-CCE4F8371F30A817	CLOUD_APPLICATION_INSTANCE-A33ADC0C699062BF
1/18/2024, 6:48:59 PM	100.127.24.163 - - [18/Jan/2024:17:48:59 +0000] "GET / HTTP/1.1" 200 659 "-" "Go-http-client/1.1"	CLOUD_APPLICATION-CCE4F8371F30A817	CLOUD_APPLICATION_INSTANCE-A33ADC0C699062BF
1/18/2024, 6:49:05 PM	10.44.72.29 - - [18/Jan/2024:17:49:05 +0000] "GET / HTTP/1.1" 200 785 "-" "ELB-HealthChecker/2.0"	CLOUD_APPLICATION-CCE4F8371F30A817	CLOUD_APPLICATION_INSTANCE-A33ADC0C699062BF
1/18/2024, 6:49:08 PM	10.44.72.254 - - [18/Jan/2024:17:49:08 +0000] "GET / HTTP/1.1" 200 785 "-" "kube-probe/1.27+"	CLOUD_APPLICATION-CCE4F8371F30A817	CLOUD_APPLICATION_INSTANCE-A33ADC0C699062BF

Tip: Platform Observability from git to prod

git commit

Backstage

Jenkins

HARBOR

Kyverno

Argo CD

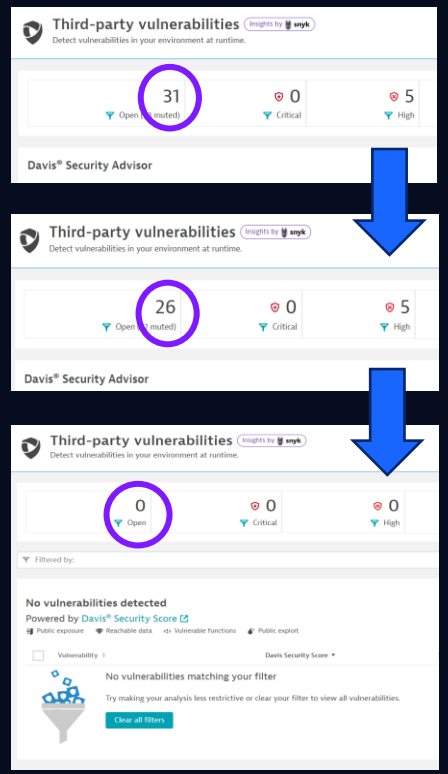
Crossplane

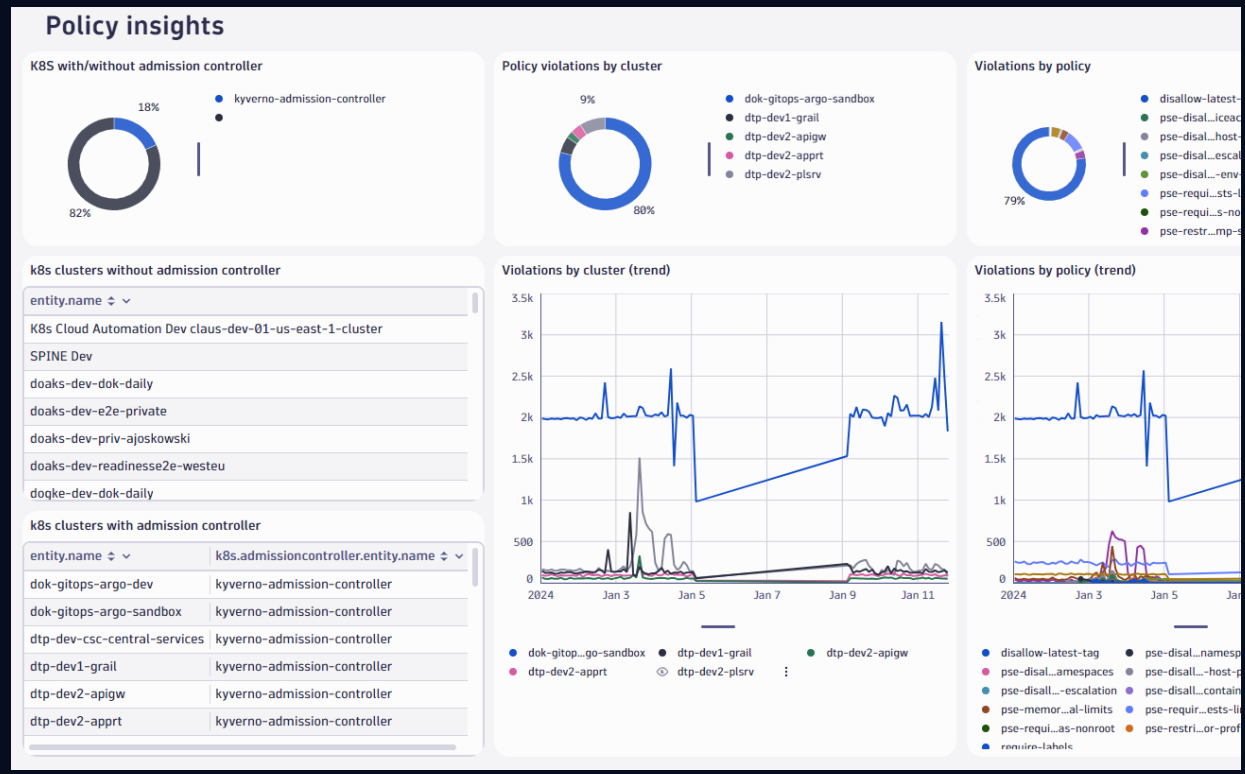
Platform Engineering: No vulnerabilities in prod? All policies enforced?

DEV

HARDE NING

PROD





Tip: Platform Observability from git to prod

git commit

Backstage

Jenkins

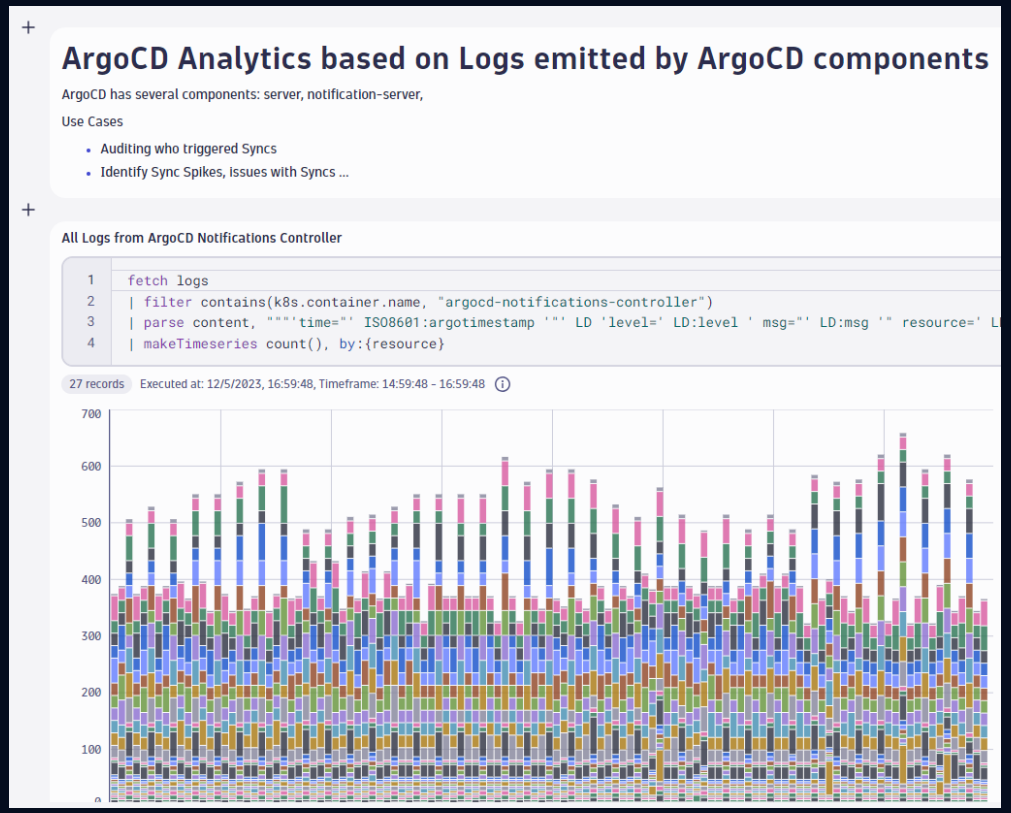
HARBOR

Kyverno

Argo CD

Crossplane

Platform Engineering: How many apps are onboarded? How well does it run?



Pipeline Success Rate

83,78%

Projects	Pipelines
1	42
Jobs	Runners
116	1

Tip: Platform Observability from git to prod

git commit

Backstage

Jenkins

HARBOR

Kyverno

Argo CD

Crossplane

Platform Engineering: Can we deploy in-time or do we have issues in stages?

From Tool Observability to Pipeline **Insights**

GitHub - Pipeline observability

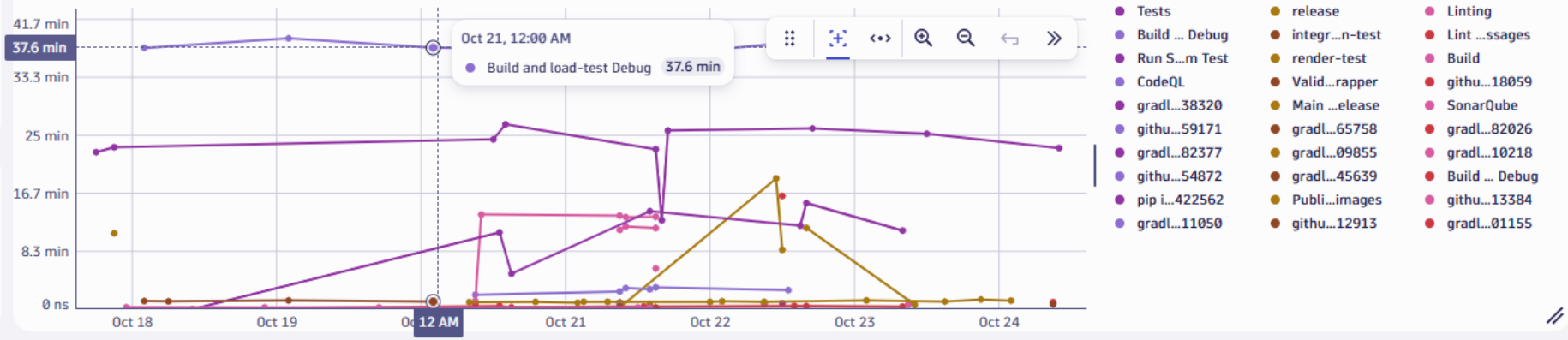
Avg. Success Rate

86.67%

Avg Execution Time

5min

Avg pipeline execution time per hour



10 pipelines with highest failure rate

name	success_rate
Build and load-test Debug	0.00%
pip in /test/load for requests - Update #905422562	0.00%
Main push build, test and publish Release	40.00%
Run Skaffold Pipeline, Deploy to Minikube Cluster and run Unguard Helm Test	50.00%
Lint Commit Messages	62.00%
render-test	76.00%
Build	90.00%
Tests	90.00%
CodeQL	92.00%

10 Slowest pipelines

name	executions	avg_time	percentile90
Build and load-test Debug	7	38.39 min	41.36 min
Tests	11	22.31 min	27.09 min
Build and test Debug	1	16.28 min	16.28 min
Build	11	12.27 min	14.77 min
SonarQube	4	11.66 min	12.42 min
release	1	10.92 min	10.92 min
Run Skaffold Pipeline, Deploy to Minikube Cluster and run Unguard Helm Test	8	9.24 min	14.50 min
Main push build, test and publish Release	5	7.27 min	19.15 min
Publish images	2	6.18 min	11.39 min
gradle in /. for org.junit.jupiter:junit-jupiter-params - Update #904910218	1	5.85 min	5.85 min

Pipeline executions

name	startTime	endTime	duration

Identifying tool optimization potential

How to improve my tooling?

Measures

System resource consumption (Bottlenecks or over provisioning)
- CPU, Memory, Disk, Network, …
- Scaling (Horizontal / Vertical)

Task Execution
- Success/Error rate on a task
- Success/Error rate overall

Task Volume
- Max, Min, Average
- Timeframe base
- Parallelism

Task Time spent
- Max, Min, Average on a Task
- Anomaly detection / Unusual deviations

Relation to other tasks
- Complexity / Effort grading

Pipeline/Workflow
- Size
- Efficiency
- Complexity
- Quality

$$\text{Average Time on Task} = \frac{\text{Average time taken to complete one task}}{\text{Total time to complete all tasks}}$$

$$\text{Error Occurence Rate} = \frac{\text{Total number of occured errors for all users}}{\text{Total number of error opportunities for all users}}$$

KPIs for every individual tool *help us verify tool performance*

SDLC Events for each Task *allow us optimize single pipelines/workflows and start with future predictions*

$$\text{Task Success Rate} = \frac{\text{Number of correctly completed tasks}}{\text{Total number of attempts}}$$

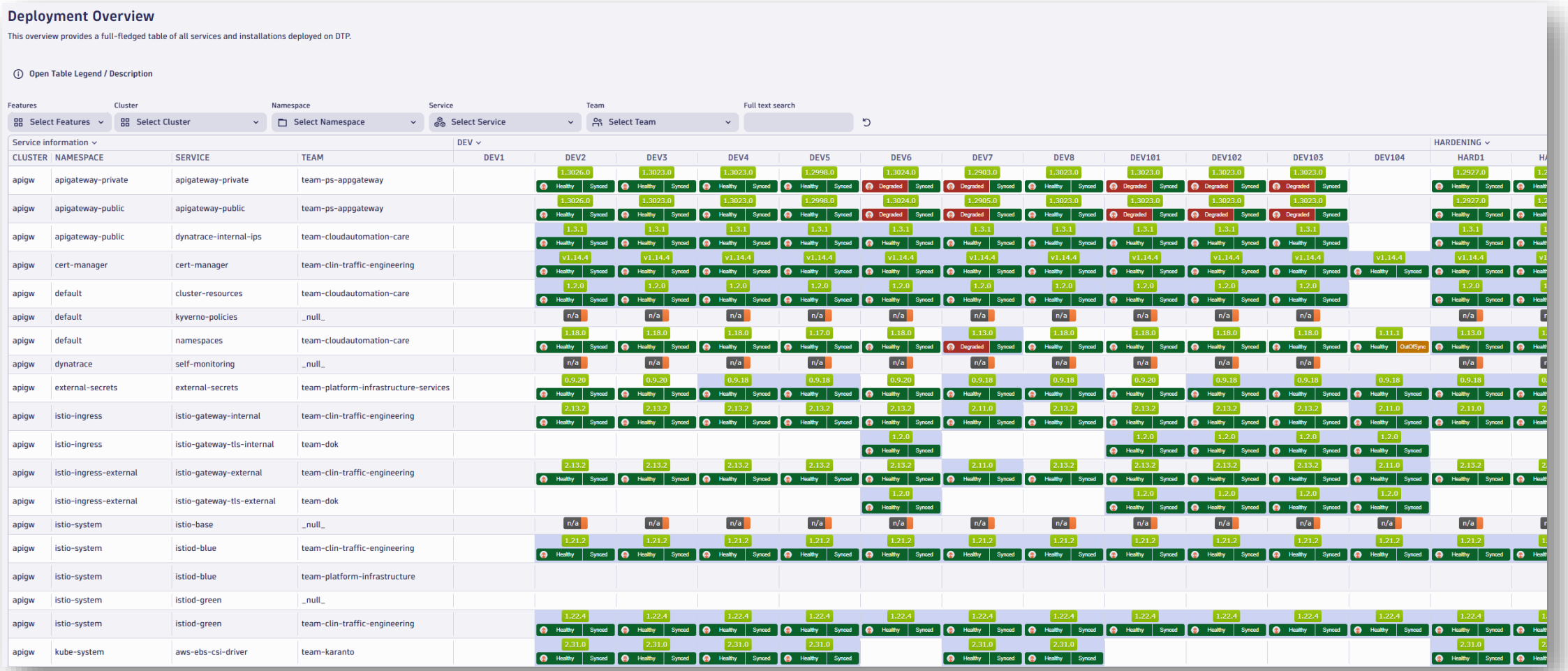
$$\text{Error Rate} = \frac{\text{Number of errors}}{\text{Total number of task attempts}}$$

Deployment/Runtime overview

What version is where deployed?

Deployment overview

*As a next step we created a **comprehensive deployment overview** in order to see **what is running where**.*

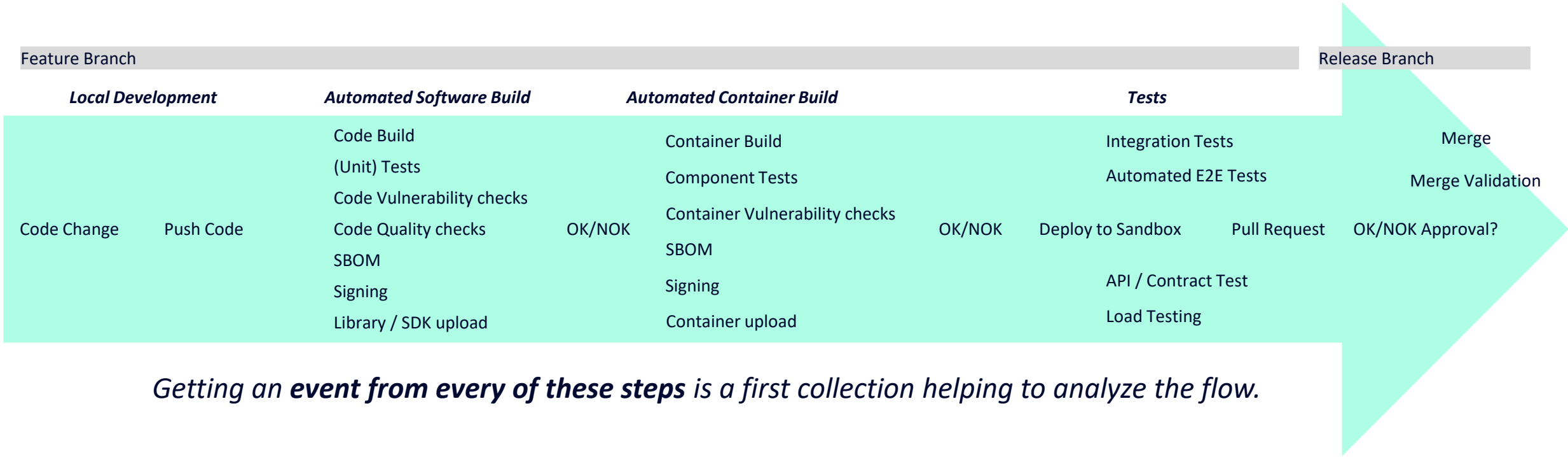


Artifact flow detection

How did the “stuff” get there?

Detecting involved tools/systems for a single artifact/code

How do you know what systems, processes and jobs are involved into your DevOps Process?

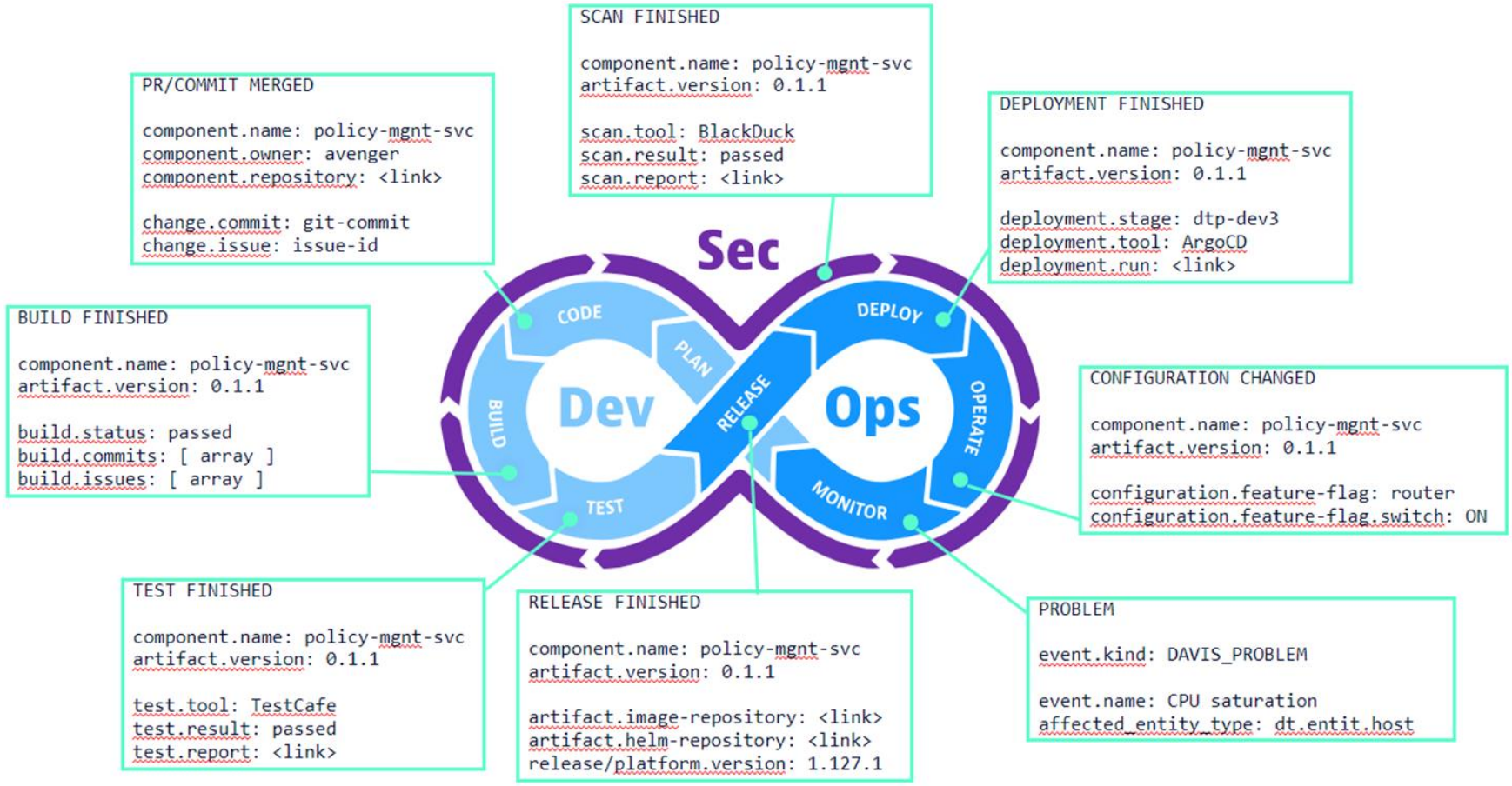


*Getting an **event from every of these steps** is a first collection helping to analyze the flow.*

As soon as you are **jumping tool boundaries** you **need a key** that individually **connects** caller with executer.

Leading to **SDLC** (Software Delivery Lifecycle Events) **Events**

*All jobs or actions need to trigger **SDLC Events with relevant information** to enrich the system observability*



We embedded SDLC Events into our Sematic Dictionary to ensure consistent usage.

Identifying flow optimization potential

Methods

Time
- End to end execution time
- Wait time / Lead time
- Transition time

Efficiency
- Amount of released/skipped versions
- Delta of sum of flow steps to overall delivery time

Human effort
- Number of human interactions vs. automated steps
- Time required for human action and complexity
- Execution/ Maintenance effort booked for flow

System complexity
- Number of tools required (end to end)
- Number of pipelines for a flow

Problems
- Issues counted end to end
- Issues during transition from one stage to the next
- Problem detection/resolution time

Testing
- Number of tests triggered
- Number of tests successfully/failed
- Test coverage

KPIs on** the specific **flows** help us to **identify** issues and **optimization potential

*Individual **SDLC Events for each Task & Flow** are needed*

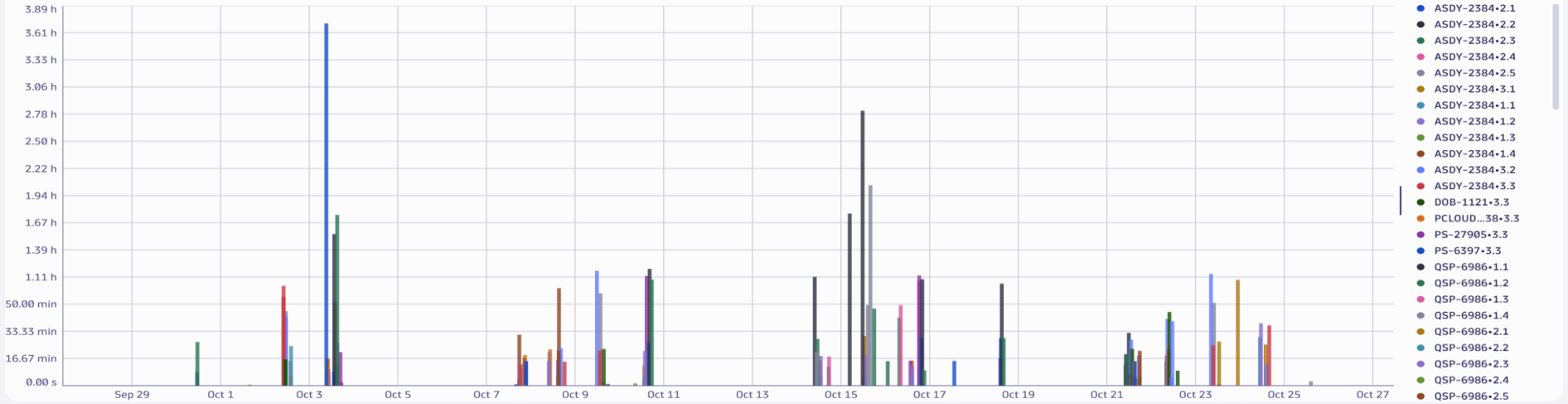
Quality Gates** help us **to ensure the KPIs** are **really meet

Deployment phases

*We built a **visualization** of our automated **rollout processes** with it's phases and **relevant measures**.*

Phased Platform Release Rollout Metrics

Elapsed time per phase



Phase	Content
Phase-1.1	ArgoCD ApplicationSets
Phase-1.2	Installation Template
Phase-1.3	Argo Workflows
Phase-1.4	DTP Orchestration Configs
Phase-2.1	Infrastructure - Base
Phase-2.2	Infrastructure - Grail
Phase-2.3	Infrastructure - ApiGW
Phase-2.4	Infrastructure - Platform Services
Phase-2.5	Infrastructure - App Runtime
Phase-3.1	ArgoCD Applications - Grail
Phase-3.2	ArgoCD Applications - ApiGW
Phase-3.3	ArgoCD Applications - Plsrv
Phase-3.4	ArgoCD Applications - AppRT

Documentation

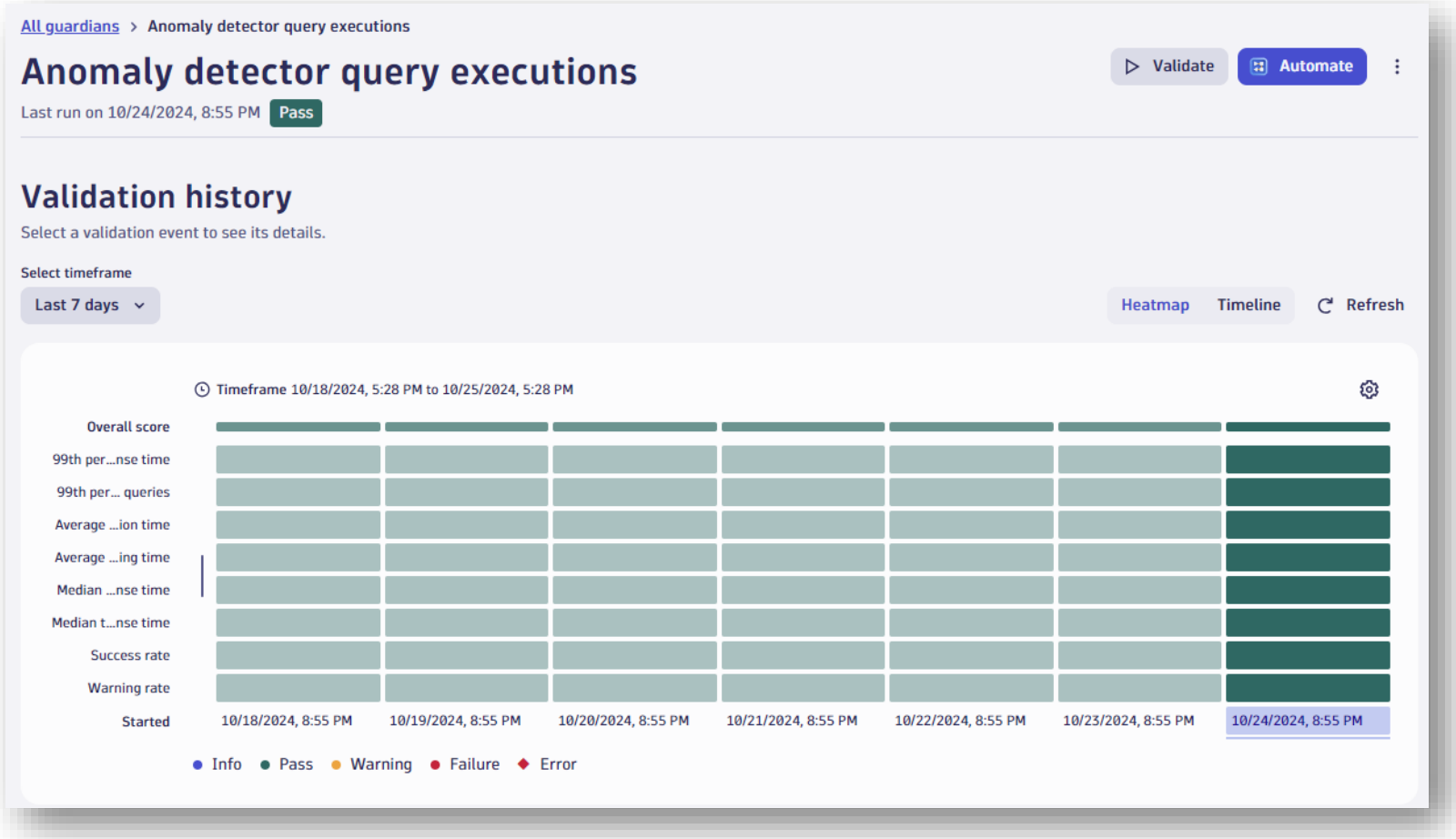
Release phases

jiraTicket	phase	status	totalDuration	executionDuration	waitDuration	start	end	runCount	failureCount	failureRate	prCount	prAvgMergeDuration	checkInstallationAvgDuration
QSP-6986	1	finished	4.72 h	6.32 min	4.61 h	10/1/2024, 10:00:36 AM	10/1/2024, 2:43:33 PM	3	0	0.00%	8	36.75 s	1.13 s
QSP-6986	2	finished	32.55 min	46.82 min	-14.27 min	10/1/2024, 2:48:04 PM	10/1/2024, 3:20:38 PM	4	2	50.00%	10	41.80 s	1.00 s
QSP-6986	3	finished	6.03 d	21.92 h	5.12 d	10/2/2024, 9:34:11 AM	10/8/2024, 10:20:10 AM	14	5	35.71%	59	43.02 s	6.89 min
	1	finished	1.89 w	11.13 min	1.89 w	10/8/2024, 11:14:50 AM	10/21/2024, 4:38:02 PM	3	missing Jira ticket	missing Jira ticket	6	7.41 min	2.57 s
	2	finished	1.88 w	1.78 h	1.87 w	10/8/2024, 2:07:04 PM	10/21/2024, 5:40:27 PM	2	missing Jira ticket	missing Jira ticket	10	39.40 s	9.85 min
	3	failed	running	19.70 h	*null*	10/9/2024, 11:19:06 AM	10/25/2024, 5:58:13 PM	10	missing Jira ticket	missing Jira ticket	178	54.94 s	6.80 min
QSP-8000	1	finished	4.12 d	7.17 min	4.11 d	10/10/2024, 8:03:49 AM	10/14/2024, 10:54:36 AM	3	0	0.00%	7	51.00 s	2.86 s
QSP-8000	2	finished	1.56 w	3.41 h	1.54 w	10/10/2024, 12:53:25 PM	10/21/2024, 11:19:47 AM	5	0	0.00%	25	40.24 s	7.35 min
QSP-8000	3	finished	1.56 w	11.80 h	1.49 w	10/10/2024, 2:15:08 PM	10/21/2024, 11:56:29 AM	13	2	15.38%	76	38.68 s	8.56 min
ASDY-2384	1	finished	1.85 d	8.33 min	1.84 d	10/21/2024, 3:53:18 PM	10/23/2024, 12:10:19 PM	6	2	33.33%	6	37.67 s	1.17 s
ASDY-2384	2	finished	2.34 h	9.80 min	2.18 h	10/23/2024, 9:54:37 AM	10/23/2024, 12:15:08 PM	3	1	33.33%	8	44.13 s	2.00 s
ASDY-2384	3	failed	running	4.25 h	*null*	10/23/2024, 12:16:33 PM	10/25/2024, 3:22:25 PM	6	5	83.33%	15	5.72 min	5.89 min
QSP-8873	1	finished	1.26 d	4.03 h	1.09 d	10/24/2024, 10:23:14 AM	10/25/2024, 4:31:37 PM	2	1	50.00%	3	38.00 s	3.00 s

*Depicts a low-level DEV Stage.

Quality Gates – Site Reliability Guardian (SRG)

*In order to ensure only software that was **properly tested** and is **adhering** to it's **non functional requirements** (NFRs) we are using **Quality Gates** to steer the flow.*



Our learnings

Conclusion

Our learnings

It is **well invested effort** to get all your **tools** and **flows observed** as you can gain significantly **improve your delivery**

You need a system that allows you to **collect** and **aggregate** all the **information**

Analysis is ideally done in **Realtime** to allow Quality Gates act instantly

You need to be able to **capture your system flows** end to end

You need to identify “keys” to **connect** the **steps** automatically

You need the **right visualizations** to give your teams **easy access** to their **relevant KPIs**

Bake the **observability as mandatory process** in your **developer platform** to ensure **standardization and correctness**

Observability is key to find the improvement areas and allow to **continuously measure and improve!**

Q & A

Q & A & Contact Details – Also find us at Booth J9!

Michael Glaess, Chief Product Architect
michael.glaess@dynatrace.com

Andreas Grabner, CNCF Ambassador & DevRel
andreas.grabner@dynatrace.com

CLOUD DONE RIGHT