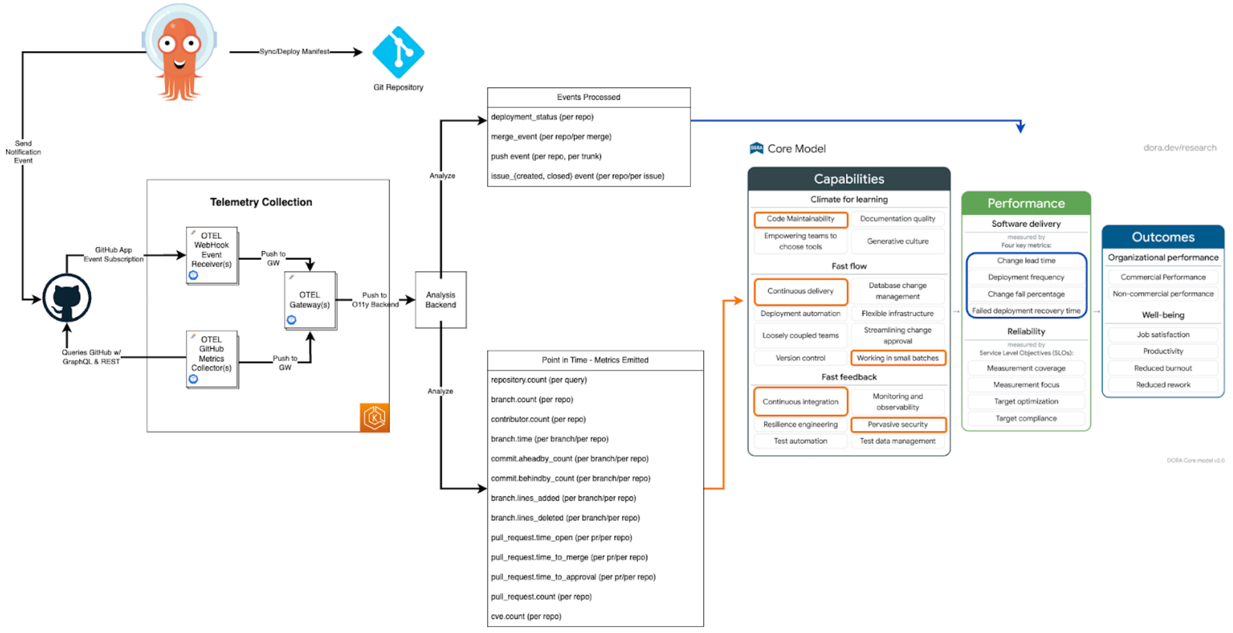
VCS Metrics and Visualization

Here's some information on visualizations created from the VCS metrics that are generated in the GitHub Receiver. Largely, the metrics defined act as leading indicators that map to DORA capabilities. Prior to the adjustments that have been made these metrics aligned closely with the Engineering Effectiveness leading indicators. These are mentioned for historical context, providing information around how these metrics are interpreted.

The following diagram is a depiction of collecting telemetry from GitHub to populate DORA Capability and Four Key metrics.


Sync/Deploy Manifest
Git Repository
Send Notification Event
Telemetry Collection
GitHub App Event Subscription
OTEL WebHook Event Receiver(s)
Push to GW
OTEL Gateway(s)
Push to O11y Backend
OTEL GitHub Metrics Collector(s)
Push to GW
Queries GitHub w/ GraphQL & REST
Analysis Backend
Analyze
Analyze
Events Processed
deployment_status (per repo)
merge_event (per repo/per merge)
push event (per repo, per trunk)
issue_{created, closed} event (per repo/per issue)
Point in Time - Metrics Emitted
repository.count (per query)
branch.count (per repo)
contributor.count (per repo)
branch.time (per branch/per repo)
commit.aheadby_count (per branch/per repo)
commit.behindby_count (per branch/per repo)
branch.lines_added (per branch/per repo)
branch.lines_deleted (per branch/per repo)
pull_request.time_open (per pr/per repo)
pull_request.time_to_merge (per pr/per repo)
pull_request.time_to_approval (per pr/per repo)
pull_request.count (per repo)
cve.count (per repo)
Core Model
dora.dev/research
Capabilities
Climate for learning
Code Maintainability
Documentation quality
Empowering teams to choose tools
Generative culture
Fast flow
Continuous delivery
Database change management
Deployment automation
Flexible infrastructure
Loosely coupled teams
Streamlining change approval
Version control
Working in small batches
Fast feedback
Continuous integration
Monitoring and observability
Resilience engineering
Pervasive security
Test automation
Test data management
Performance
Software delivery
measured by Four key metrics:
Change lead time
Deployment frequency
Change fail percentage
Failed deployment recovery time
Reliability
measured by Service Level Objectives (SLOs):
Measurement coverage
Measurement focus
Target optimization
Target compliance
Outcomes
Organizational performance
Commercial Performance
Non-commercial performance
Well-being
Job satisfaction
Productivity
Reduced burnout
Reduced rework
DORA Core model v2.0

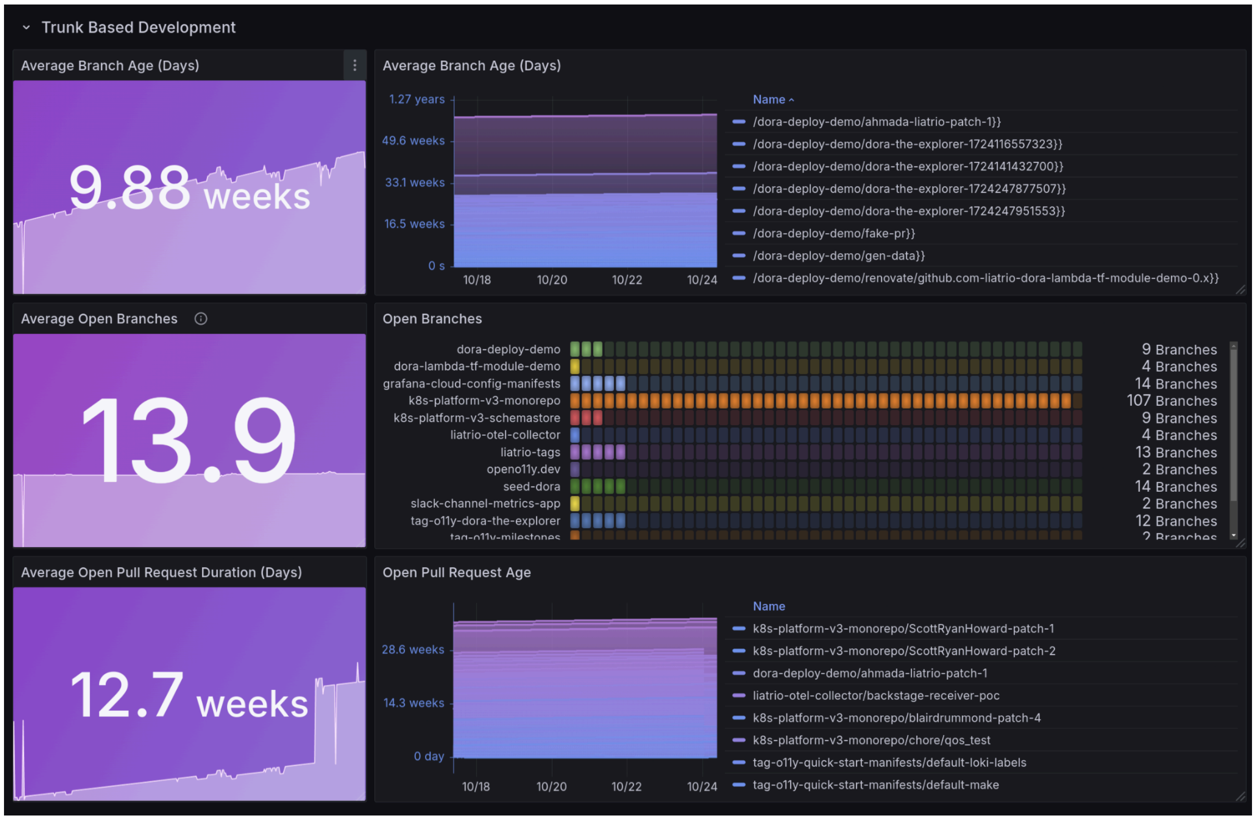
These VCS metrics are point-in-time and scraped from the API. This gives a view of the current state. From there, events could be used to continue adding or removing from the metrics. When developing these metrics, we thought about how they would be used to tell a story. The story is human. Consider the following hypothetical scenario.

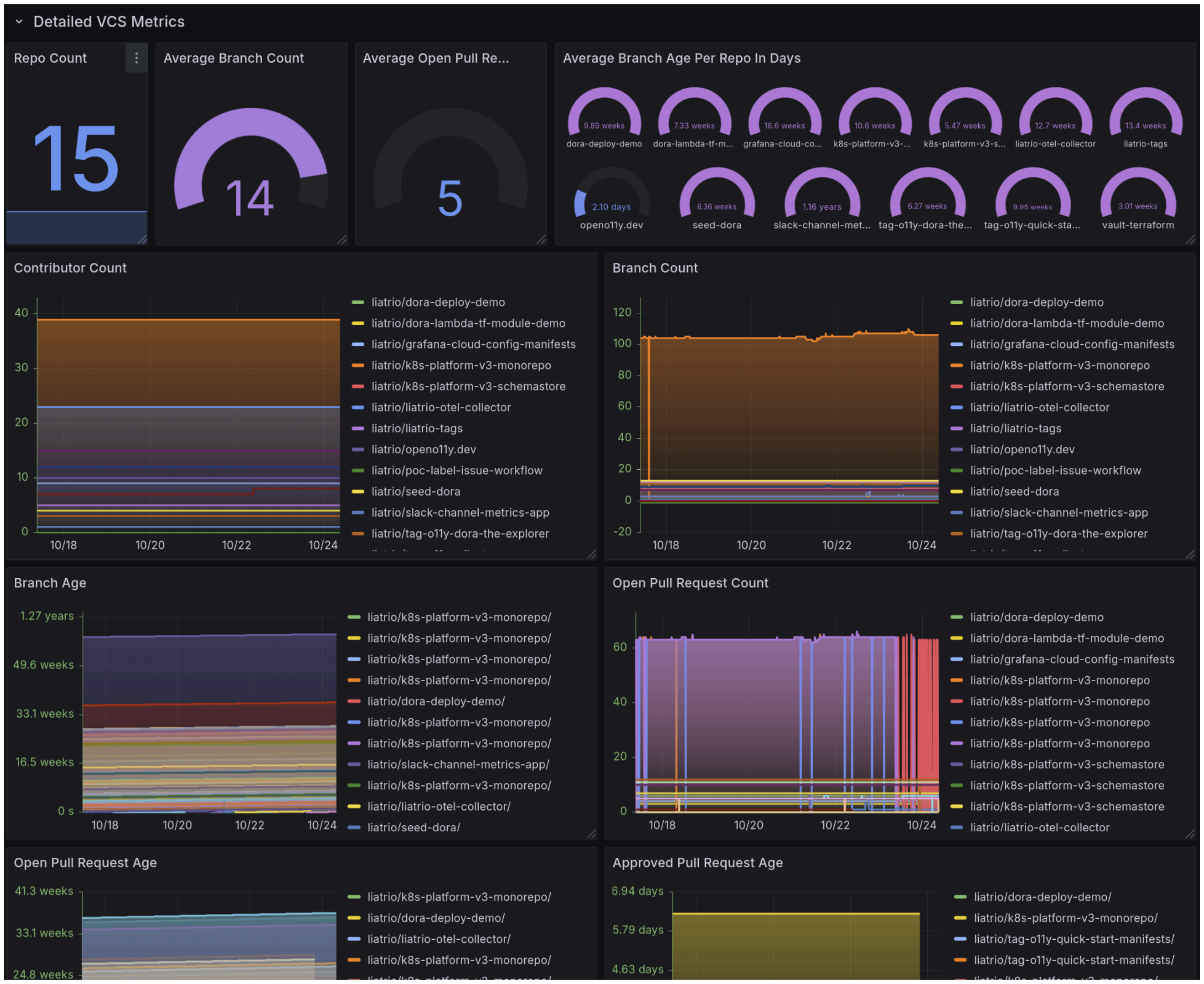
Technical leader Melissa noticed that her team had what felt like a high number of open pull requests (vcs.change.count && vcs.change.state = open) across services her team owns. She logically went to see what the average open time for pull requests was. From there she deduced that pull requests were sitting open for weeks at a time, contributing to the high open count. Why were pull requests open for so long? She knew that her team was small, consisting of only four engineers including herself. However, her team of four owned twenty-two repositories, consisting of six services. She realized this was a lot to maintain, but it didn't paint the entire picture. Her team certainly received inner-source contributions from other teams across the organization. Milissa then took a look at the average line addition and removal accounts for the various branches that existed. There she realized that each pull request included too many changes, making the pull requests hard to review. All these data points allowed her to reason that there was change needed to reduce cognitive load on the team, and accelerate delivery of value.

This was the original thinking behind the value of the metrics, measuring aspects of the DORA capabilities and providing the fundamental data points to create aggregations against. Each datapoint, often a simple gauge. This allows the user to create simple, but arguably effective aggregations through queries like (in prometheus as the example):

- `This query gets the average branch age for repositories owned by a team.`
    - `avg by(team_name) (vcs_repository_ref_time_seconds{organization_name=~"$organization", repository_name=~"$repo", team_name=~"$team"}) / 86400`
- `This query gets the average branch count across repositories owned by a team.`
    - `avg by(team_name) (vcs_repository_ref_count{organization_name=~"$organization", repository_name=~"$repo", team_name=~"$team"})`
- `This query gets the average pull requests open time for repositories by team.`
    - `avg by(team_name) (vcs_repository_change_time_open_seconds{organization_name=~"$organization", team_name=~"$team", repository_name=~"$repo"}) / 86400`
- `This query gets the average branch time by repository.`
    - `avg by(repository_name) (vcs_repository_ref_time_seconds{organization_name="$organization", repository_name=~"$repo", team_name=~"$team"}) / 86400`

The resulting dashboard would like as follows (taken from the public DORA Capabilities Dashboard):

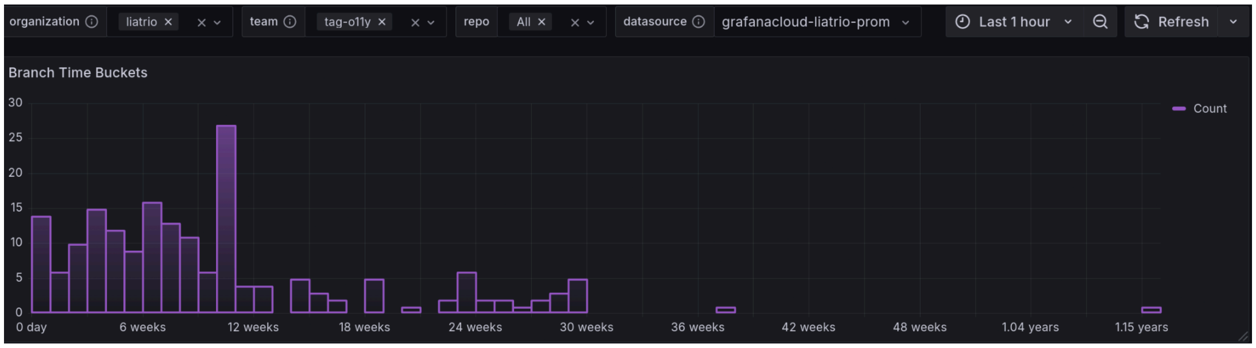

Trunk Based Development
Average Branch Age (Days)
9.88 weeks
Average Branch Age (Days)
1.27 years
49.6 weeks
33.1 weeks
16.5 weeks
0 s
10/18
10/20
10/22
10/24
Name
/dora-deploy-demo/ahmada-liatrio-patch-1}}
/dora-deploy-demo/dora-the-explorer-1724116557323}}
/dora-deploy-demo/dora-the-explorer-1724141432700}}
/dora-deploy-demo/dora-the-explorer-1724247877507}}
/dora-deploy-demo/dora-the-explorer-1724247951553}}
/dora-deploy-demo/fake-pr}}
/dora-deploy-demo/gen-data}}
/dora-deploy-demo/renovate/github.com-liatrio-dora-lambda-tf-module-demo-0.x}}
Average Open Branches
13.9
Open Branches
dora-deploy-demo
dora-lambda-tf-module-demo
grafana-cloud-config-manifests
k8s-platform-v3-monorepo
k8s-platform-v3-schemastore
liatrio-otel-collector
liatrio-tags
openo11y.dev
seed-dora
slack-channel-metrics-app
tag-o11y-dora-the-explorer
9 Branches
4 Branches
14 Branches
107 Branches
9 Branches
4 Branches
13 Branches
2 Branches
14 Branches
2 Branches
12 Branches
Average Open Pull Request Duration (Days)
12.7 weeks
Open Pull Request Age
28.6 weeks
14.3 weeks
0 day
10/18
10/20
10/22
10/24
Name
k8s-platform-v3-monorepo/ScottRyanHoward-patch-1
k8s-platform-v3-monorepo/ScottRyanHoward-patch-2
dora-deploy-demo/ahmada-liatrio-patch-1
liatrio-otel-collector/backstage-receiver-poc
k8s-platform-v3-monorepo/blairdrummond-patch-4
k8s-platform-v3-monorepo/chore/qos_test
tag-o11y-quick-start-manifests/default-loki-labels
tag-o11y-quick-start-manifests/default-make


Detailed VCS Metrics
Repo Count
15
Average Branch Count
14
Average Open Pull Re...
5
Average Branch Age Per Repo In Days
9.89 weeks dora-deploy-demo
7.33 weeks dora-lambda-tf-m...
16.6 weeks grafana-cloud-co...
10.6 weeks k8s-platform-v3-...
5.47 weeks k8s-platform-v3-s...
12.7 weeks liatrio-otel-collector
13.4 weeks liatrio-tags
2.10 days openo11y.dev
6.36 weeks seed-dora
1.16 years slack-channel-met...
6.27 weeks tag-o11y-dora-the...
9.95 weeks tag-o11y-quick-sta...
3.01 weeks vault-terraform
Contributor Count
Branch Count
Branch Age
Open Pull Request Count
Open Pull Request Age
Approved Pull Request Age
liatrio/dora-deploy-demo
liatrio/dora-lambda-tf-module-demo
liatrio/grafana-cloud-config-manifests
liatrio/k8s-platform-v3-monorepo
liatrio/k8s-platform-v3-schemastore
liatrio/liatrio-otel-collector
liatrio/liatrio-tags
liatrio/openo11y.dev
liatrio/poc-label-issue-workflow
liatrio/seed-dora
liatrio/slack-channel-metrics-app
liatrio/tag-o11y-dora-the-explorer
10/18
10/20
10/22
10/24
1.27 years
49.6 weeks
33.1 weeks
16.5 weeks
0 s
41.3 weeks
24.8 weeks
6.94 days
5.79 days
4.63 days
liatrio/k8s-platform-v3-monorepo/
liatrio/dora-deploy-demo/
liatrio/slack-channel-metrics-app/
liatrio/liatrio-otel-collector/
liatrio/seed-dora/
liatrio/tag-o11y-quick-start-manifests/

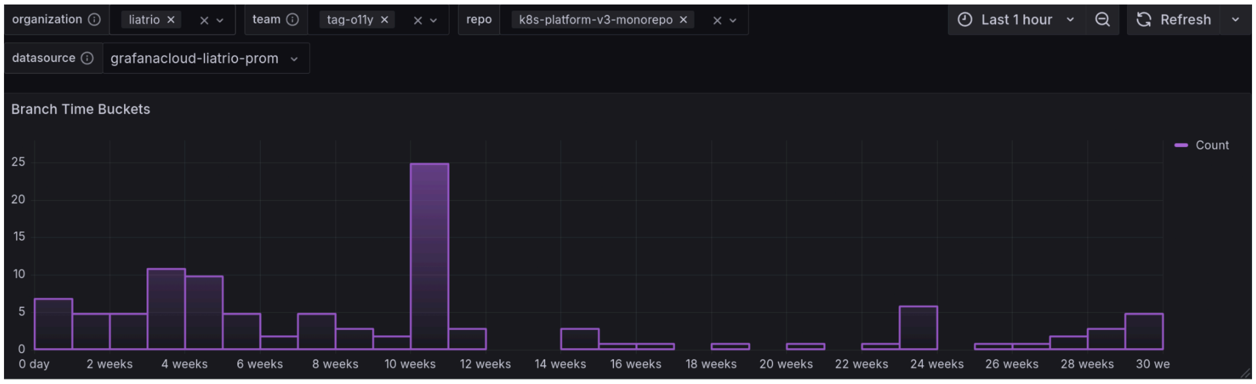
Histograms can be beneficial to detect outliers and distributions. From the fundamental gauge metric of time in seconds, one can create a simple histogram fairly easily. This may feel different, depending on the analysis backend, but in the examples above we're using Grafana Cloud, where the metrics are stored in Mimir (Prometheus). The query itself stays the same.

```
vcs_repository_ref_time_seconds{organization_name=~"$organization",
repository_name=~"$repo", team_name="tag-o11y"} / 86400
```

For the buckets, I created a bucket size of seven, and the unit is represented in days. Therefore each bucket is a week. I ran one transformation on the data which was a `reduce` function to reduce the series to rows, taking the max of the value. This way I'm not counting multiples of the same metrics. The rendering of the visualization is seen below.



Now, because this is all based on attributes, you can pair down to see distributions by repository simply changing the variable.



To get the resolution though and analyze further, one would want to dive deeper by looking at the fundamental telemetry prior to aggregation.

Summary

Based on the fundamental metric types, it’s easy enough to query for the information you want, and still render distributions. The dashboards shown here are publicly available through the Grafana Dashboard marketplace and get updated as the metric conventions are updated.