Who Can Approve Kubernetes

Stated, held and exercised approval authority in a large open-source project

Jordan Myska Allen
UpTrust
Austin, Texas, USA
jordan@uptrusthq.com

Brian Raszap
UpTrust
Tel Aviv, Israel
br@uptrusthq.com

Emin Göbütoğlu
UpTrust
Ankara, Türkiye
emingbt@uptrusthq.com

Abstract

Stated, held and exercised authority differ per person in Kubernetes. Measured over the same pull requests, a broad right to approve does not predict heavy or light exercising of that right. Also, stated Special Interest Group leadership (SIG, the project's departments) does not necessarily match the SIG where a person approves the most.

Seven people were authorized to sign off on almost any change to Kubernetes's core repository; one of them had the authority on 98% of the (publicly recorded) merged changes. In practice it took four people's sign-offs to cover half of those changes; one person with approval authority on 83% of the changes only signed off on a single one in eight months. The person who signs off most in each Special Interest Group is usually an official leader of a *different* SIG.

The measurements come from three public, machine-readable records: the `sigs.yaml` leadership roster, per-directory `OWNERS` permission files, and the `/approve` commands themselves, over eight months of the GitHub event archive (2023-10 to 2024-05) and `kubernetes/kubernetes` at pinned commits; eligibility is computed per pull request against the `OWNERS` state at merge time. Every figure reproduces from public data with published code, with sensitivity reported to attribution channel, multi-label credit, roster version, command type, threshold and `OWNERS` snapshot.

CCS Concepts

• **Software and its engineering → Collaboration in software development**; *Software version control.*

Keywords

Kubernetes, OWNERS, code review, approval authority, mining software repositories, open-source governance, Prow

1 Introduction

Research on formal versus informal structure is old [8], and the claim that system structure copies organizational communication structure is older still [3]. The software-engineering instance of the first is well known: a maintainer file says one thing, the revision history says another. Kubernetes is unusual because it keeps three such records, and all three are public, complete, machine-readable at the level of the individual act, and appear intentionally separated.

- **Leadership named (stated).** `kubernetes/community` publishes `sigs.yaml`, naming the chairs and technical leads of every SIG, together with working groups and committees.
- **Permissions (held).** A directory may have an `OWNERS` file listing approvers. The project's automation, Prow, requires an `/approve` from an applicable approver, plus an `/lgtm`, before a change merges, so authority is path dependent, rather than title dependent.
- **Behavior (exercised).** Both commands are ordinary public comments, so every act of approval is attributable and archived.

Comparing official titles against who can approve is not that interesting here, because the project already documents them as different things: the OWNERS documentation is explicit that approval is delegated per path, and the contributor guide states plainly that "there are a handful of people who can approve changes across large portions of the repository."

We therefore ask four narrower questions:

1. **How concentrated is the permission pool itself?** If a handful of people can approve across large portions of the repository, how large, per person?
2. **How concentrated is the exercise of that permission?**
3. **Where both can be measured over one population of pull requests, how does held permission compare with exercised permission?**
4. **How far apart are the leadership record and the approval stream, once the roster is versioned and the two commands are separated?**

This is a single-project case study. We measure a single project's governance records that publishes all three of them, and we make no claim that the distributions generalize: even within free and open-source software, centralization varies too widely across projects for one project's shape to stand in for the rest [4]. Where a claim depends on Kubernetes-specific machinery (Prow commands, `OWNERS` inheritance, SIG labels) we say so.

Section 3 states which results we believe are new.

2 Background: the three records

`sigs.yaml` is a governance document that declares Chairs who run meetings, charters and escalation; and technical leads who set direction. Code approval rights are separate:

`OWNERS` files are the approval permission system, which include:

- an approver listed in directory D's `OWNERS` covers D and all descendants (inheriting subdirectory approvals);
- a descendant setting `options.no_parent_owners: true` cuts that inheritance at itself (46 such directories at v1.30.0);
- `filters` maps a regular expression to its own approver list; a filter of `.*` covers the whole directory, a narrower one covers only some files;
- `emeritus_approvers` are former approvers and confer nothing;

- alias names expand through `OWNERS_ALIASES` (71 aliases at v1.30.0).

The third record, the approval commands themselves, needs no background beyond Section 4.3: they are comments.

3 Previous work and what is new here

Concentration in Kubernetes review has been measured. Mirsaeedi and Rigby [10] compute a Gini coefficient over reviewer workload on a dataset that includes Kubernetes, reporting that the top fifth of reviewers perform 75% to 84% of reviews. Their reviewer events combine comments, change requests and approvals, resulting in one figure per project.

Ownership per module per person has been built. Thongtanunam et al. [11] extend authorship-based ownership with review-based ownership at directory granularity, and report that most contributors to a module participate only through review. Their signal is review comments rather than approval votes, and their projects are Qt and OpenStack.

The stated-versus-revealed comparison has been done at scale. Lulla et al. [9] compare declared `CODEOWNERS` against behavior across 222 repositories and 844,492 pull requests, finding that 79% of declared *individual* code owners, as distinct from team owners, are not among their repository's top hundred committers. `CODEOWNERS` is the direct analogue of `OWNERS`, but the unit is the pull request and concentration over the mapping is not measured. Kubernetes does not appear in their sample; since it grants approval through Prow `OWNERS` files rather than GitHub `CODEOWNERS`, we take it to fall outside their selection criteria (inferred, not stated by them).

Counting approvals per person is also pre-existing. CNCF's public DevStats dashboard already counts `/approve` comments per person from the same archive we use. It does not attribute those approvals to SIGs inside the main repository. DevStats organizes activity by *repository group*: a SIG's group is the set of separate repositories that SIG owns. Most Kubernetes code lives in the single `kubernetes/kubernetes` repository, and a whole-repository grouping does not say which SIG a change inside it belongs to; we use the `sig/*` labels for that (Section 4.6). DevStats also reports each group on its own and does not report how one person's approvals spread across SIGs. We look at this question ing Section 6.4.

Adjacent Kubernetes work. Aufiero et al. [1] build a co-commenter network over 46,768 issues and 339,332 comments spanning eleven years, and report concentration in it: a Gini coefficient of 0.69, with 7.74% of contributors holding half of all connections. Their unit of structure is a set of 602 emergent bipartite modules found by Leiden community detection, so their domains have no owners by construction, but they do test those modules against the project's own labels (SIG labels among them) and propose that comparison as a way "to evaluate whether formal team organisation, working groups, or ownership policies reflect how contributors actually coordinate." We ask a similar question using a different record. Their corpus is issues and issue comments, with no pull requests, reviews, approval commands or `OWNERS` data, and they state the consequence in their own threats to validity: focusing on issues rather than pull requests "may underrepresent review activity." The present paper measures the activity they identify as underrepresented over the permission system that governs it.

Vaccargiu et al. [12], from the same group, do work on pull requests, comparing `area/` labels against file diffs across 18,020 Kubernetes pull requests and over a million review comments. The subject is label hygiene rather than authority, and it likewise does not touch approval rights.

Bus-factor literature measures something else. Truck-factor methods [2] estimate how many people a project cannot afford to lose from file authorship in commit history, and report that 65% of the 133 projects studied have a truck factor of two or less. Kubernetes is in none of the canonical datasets. Cury and Avelino [5] compute a truck factor per folder, which is the nearest published shape to ours, still from authorship. Jabrayilzade et al. [7] is the one bus-factor method that admits review data, combining it with version control and meetings into a project-level scalar.

What is new. Three results, none of which we found published for any project. First, the permission system reduced to a per-person share: one person can approve 98.6% of the core repository's owner-ruled directories and 98.4% of its merged traffic (Sections 6.1 and 6.3). The prior work above measures who *reviewed* or *committed*; none of it measures what a person is *allowed* to approve. Second, eligibility and exercise measured per person over the same pull requests (Section 6.3): two broad approvers approve half of the core's merged traffic while a third with comparable permission approved one pull request, so held authority does not predict exercised authority. Lulla et al. [9] compare declared owners against committers, but per repository and without concentration; nobody has compared who *could* approve each change against who *did*. Third, that the leadership roster misplaces rather than misses the busiest approvers (Section 6.4). Neither of the nearest published Kubernetes studies touches approval rights: Vaccargiu et al. [12] studies label hygiene, and Aufiero et al. [1]'s dataset contains no pull request or review data. If any of the three has been published, one citation refutes the claim.

4 Data

4.1 Sources

All are public. The first three require no account; the fourth requires a free GitHub token.

1. **The GitHub event archive** via the ClickHouse public playground (`play.clickhouse.com`, table `github_events`), for events on repositories under the `kubernetes` and `kubernetes-sigs` organizations.
2. **`kubernetes/kubernetes` itself**, at pinned tags for the coverage table and across its first-parent history for the eligibility analysis, for `OWNERS` files.
3. **`sigs.yaml`** from `kubernetes/community`, at every in-window revision.
4. **The GitHub API**, for the changed-file list and merge timestamp of each core-repository pull request in the eligibility analysis (Section 5.4).

Mining GitHub's public event stream at scale is standard practice [6].

4.2 The permission register, at pinned commits

We analyze v1.30.0 (`7c48c2bd72b9bf5c44d21d7338cc7bea77d0ad2a`, 2024-04-17, inside the activity window) and v1.29.0 (`3f7a50f38688eb332e2a1b013678c6435d539ae6`, 2023-12-13) for stability. At v1.30.0: 516 `OWNERS` files, 71 aliases, 46 directories cutting parent inheritance, 5,406 directories in the tree.

Our primary denominator is **owner-file directories**: the 501 directories, excluding vendored trees, that carry an `OWNERS` file. A share of these is the share of the places the project wrote an ownership rule that a person can approve in. We also report shares of **all directories**, the 4,063 non-vendored directories, a denominator dominated by generated `staging/` trees and by directories the project never wrote a rule for. Vendored code (`vendor/`, `third_party/`) is excluded from the coverage denominators; `vendor/` alone is over 1,300 directories assigned wholesale to a dependency alias, and including it changes the ordering for reasons unrelated to authority over Kubernetes. (The per-pull-request eligibility analysis of Section 6.3 evaluates vendored files, because Prow does.)

4.3 The behavioral record

Kubernetes approvals are Prow slash commands and most of them are typed as ordinary pull-request comments. We parse `IssueCommentEvent` bodies with line-anchored patterns: `(?m)^/approve(\s|$)` and `(?m)^/lgtm(\s|$)`, with `\s+cancel` variants. Bots are excluded on both sides, as approvers and as pull-request authors, by actor suffix (`[bot]`, `-bot`, `-robot`), which also excludes the notifier bot whose boilerplate a naive substring match would count as `/approve`. Since a name rule can miss a bot with an unlucky name, we also checked each login in a reported table against GitHub, in two separate AI passes, and found no automation accounts.

We parse all three kinds of GitHub event commands: plain pull-request comments (`IssueCommentEvent`), review bodies (`PullRequestReviewEvent`), and inline review comments (`PullRequestReviewCommentEvent`). A fourth channel has no command text: for `kubernetes/kubernetes`, whose Prow configuration leaves `ignore_review_state` unset, a GitHub review in the *Approved* state counts as an approval if the reviewer is an eligible approver. Any reviewer can press that button, and eligibility can be checked only where `OWNERS` files are parsed, so *Approved*-state reviews are reported as a variant in Sections 6.3 and 7 and are not in the headline counts. Table 1 gives the volume in each channel.

Authorship is resolved by joining each command comment to the `PullRequestEvent action='opened'` row on `(repo_name, number)`, with a six-month lookback, because the comment row's creator field is not the pull-request author.

Retraction. Per `(approver, pull request, command kind)` the chronologically last command decides, across all three channels: a trailing cancel retracts the act. There are 210 cancels in the corpus, 0.52% of raw command rows. A cancel is treated as retraction; manual inspection shows cancels are procedural.

Self-approvals are dropped.

4.4 Corpus shape

4.5 Window provenance and its limit

The playground's Kubernetes coverage table starts **2023-01-13** and collapses after 2024-05 (42,786 `IssueCommentEvent` rows in May 2024, 4,912 in June). Every window we test is inside this 17-month era.

4.6 SIG labels are a per-repository switch

SIG attribution uses the project's automated `sig/*` labels. Labeling is nearly all-or-nothing by repository: twelve core repositories are about 95% labeled (97.8% in `kubernetes/kubernetes`, and between 78% and 100% for every approver there with at least 20 acts), while the satellite repositories are near 0% (150 of 178 repositories are under 5% labeled; `cluster-api` is 0.0% over 2,141 acts). Overall, 36.5% of raw command rows carry a label, but those rows are the core repositories. A chi-squared test over the 97 repositories with at least 30 acts gives 36,202 on 96 degrees of freedom, and the rank correlation of approver frequency between the labeled and unlabeled rows is +0.12.

The per-SIG results therefore describe the Kubernetes core repositories and say nothing about satellite review, which is where the approver with the most pooled `/approve` and `/lgtm` acts in the corpus—sbueringer (1,595 acts, 6.5% labeled)—does almost all of that approving.

4.7 The leadership register must be versioned

A single late snapshot of `sigs.yaml` is the wrong roster for a historical window: leaders rotate, so anyone who left reads as an active non-leader and anyone who joined later reads as an inactive leader. We therefore take the union of all 114 in-window revisions of `sigs.yaml` on the main branch, and we include the Steering Committee and working-group chairs, which a chairs-and-technical-leads parse omits.

The snapshot our own first attempt used, `kubernetes/community` at `e210e578` (2026-07-10, 25 months after this window closes), carries 116 chair and technical-lead role-rows across the 24 SIGs, against 126 in the in-window union; 56 of its 116 rows disagree with the window. Restricted to the 19 SIGs that pass the traffic gate defined in Section 5.3, the snapshot carries 98 role-rows: 22 of them are seats not yet granted during the window, and it omits 36 seats that were held in-window — 58 discrepancies against 97 in-window person-seats. Omitting the Steering Committee and working groups compounds this: one person held a Steering seat throughout the window and another chaired a working group inside it, and both read as holding no position at all under a chairs-and-leads-only parse.

5 Method

5.1 Held authority: per-person approval coverage

For each directory we walk the `OWNERS` chain from that directory upwards, stopping at a cut, and union the applicable approvers, expanding aliases and ignoring emeritus entries. An approver named only under a narrow `filters` pattern is not credited with the directory; this does not change anything because because every filter

Table 1: Corpus shape. The approver rows count different sets of people — some raw, some surviving, some `/approve` only — so check a row's label before comparing it with another row.

Quantity	Value
Window	2023-10-01 to 2024-06-01
Raw command rows parsed (issue comments / review bodies / review comments)	40,640 (25,963 / 14,537 / 140)
Approved-state reviews with no command text (the Section 4.3 variant)	8,554
Surviving approval acts	27,226
Repositories with at least three attributable acts	165
Contributors (approved a PR or authored an approved one)	4,668
Distinct approvers, raw	793
Distinct approvers with a surviving act	748
Distinct approvers who ever issued `/approve`	460
Distinct approvers with a surviving `/approve` act	430
SIGs in the leadership register	24
Leadership role-rows, in-window union (Section 4.7)	126 (111 distinct person-seats)
`kubernetes/kubernetes` pull requests merged in window (GitHub)	1,984
… with any surviving `/approve`	1,588
… with a surviving, author-attributable `/approve`	1,458
… of those, merged (the Section 6.3 population)	1,418
Surviving `/approve` acts on that population	1,604

that grants approval is `.*` at both pinned refs. A person's coverage is the number of directories they can approve in, divided by the denominator. The implementation is published (Section 9).

5.2 Exercised authority (aka why there's a range instead of one number)

Over surviving `/approve` acts, we compute per-person pull-request coverage, cumulative shares for the top k approvers, a Gini coefficient over per-approver act counts, and bounds on the smallest set of people whose approvals cover half of the approved pull requests. Greedy selection (repeatedly take the approver adding the most uncovered pull requests) yields a cover and therefore an **upper** bound. For a **lower** bound we use the fact that a union cannot exceed a sum: if the k largest individual coverages sum to less than half the approved pull requests, no set of k people can cover half. Reporting both bounds avoids the common error of treating a greedy result as a minimum.

5.3 Registers compared, per SIG

For each SIG we rank contributors by distinct pull requests carrying that SIG's label on which they left a surviving command, breaking ties by count then login. We report `/approve` alone as the primary measure, because that is the OWNERS-gated power, and the pooled `/approve` + `/lgtm` measure as a variant. Every figure states which it uses. SIGs are gated at 50 labeled pull requests in the window, which admits 19 of 24; we publish the 24-SIG figures alongside, since we cannot reconstruct whether the threshold predates the first results.

5.4 Eligible approvers, per pull request

For the core repository we fix one population and measure entitlement and exercise over it. The population is every `kubernetes/kubernetes` pull request with a surviving, author-attributable `/approve` in the window that was subsequently merged: 1,418 of 1,458 (40 were approved but closed unmerged). For each, we fetch its changed file paths and merge timestamp from the GitHub API, take the `OWNERS` state of the default branch (`master`) on its first-parent history immediately before that timestamp — the state Prow evaluates — and compute two sets under the semantics of Section 5.1, evaluating every changed file including vendored ones. The **sole-eligible set** is the people whose approval alone covers every changed file; the **union-eligible set** is the people who could approve at least one of them. Sole eligibility is the primary measure, because it is the set from which a single human act can close Prow's gate. Sixty-five `OWNERS` states occur across the population. The engine is a second implementation of the Section 5.1 semantics and, pointed at `7c48c2bd`, reproduces Table 2 (Section 9).

Table 2: Approval coverage per person, unweighted, over the 501 directories that carry an `OWNERS` file at v1.30.0 (`7c48c2bd`), vendored trees excluded, narrow filters not credited, emeritus entries not credited. Shares are of directories, not of files, churn or pull-request traffic.

#	approver	directories	share
1	liggitt	494	98.6%
2	thockin	468	93.4%
3	smarterclayton	463	92.4%
4	dims	431	86.0%
5	wojtek-t	422	84.2%
6	dchen1107	397	79.2%
7	deads2k	369	73.7%
8	mikedanese	221	44.1%

Nothing in this paper uses a trust-propagation model, a clustering algorithm, or any non-public computation.

6 Results

6.1 Held authority is very concentrated

Seven people clear half, and the eighth is well below it, so the count is not knife-edge. **148 people hold an approval right anywhere.** Using all 4,063 non-vendored directories as the denominator instead, eight people clear half and the top share is 99.6%. At v1.29.0 the top share is again 98.6% and seven people again clear half; no

coverage share moves by more than 1.7 percentage points, and the top seven are the same people in the same order.

These shares are not a restatement of the root approver list. The obvious objection is that a root `OWNERS` file exists, so of course somebody covers everything, and the table merely expands root aliases. The same ref refutes this. First, root approval is deliberately non-recursive: 46 directories set `no_parent_owners`, and 16 of those are top level, including every directory holding the bulk of the code (`pkg`, `cmd`, `staging`, `plugin`, `test`, `api`, `build`, `cluster`, `hack`). `pkg/OWNERS` carries the comment `# make root approval non-recursive` verbatim. Second, the root approver set is small and mostly disjoint from the table above: root grants approval through a `.*` filter to nine people (bentheelder, cblecker, derekwaynecarr, dims, johnbelamaric, liggitt, soltysh, sttts, thockin), of whom only three appear among the seven; three others in the seven (smarterclayton, wojtek-t, dchen1107) are listed at root as `emeritus_approvers` and hold no current root authority at all, and deads2k is not in the root file in any capacity. Third, the coverage arrives by explicit relisting below the cut: the six people of Table 2 other than deads2k are named individually, as literal handles rather than through an alias, in the approvers list of `pkg`, `cmd`, `staging`, `plugin` and `test` — in `pkg`, `cmd`, `staging` and `plugin` among only six or seven approvers in total, and in `test` within a larger list of 27.

So the mechanism is not inheritance from the top. The project cut root approval off from the code on purpose, and the same people appear immediately below each cut. We describe what the files do and take no position on why they were written that way; keeping a person's approval where their work is would be an ordinary reason. For the same reason we do not read these shares as a governance failure.

This is the approval right only: Prow also requires an `/lgtm`, so it is not a count of who can merge unilaterally. And it is an unweighted directory count, not code volume, traffic, or risk; Section 6.3 tests the traffic objection directly.

6.2 Across the ecosystem, exercised authority is much less concentrated

Over surviving `/approve` acts in the window, across both organizations: 9,601 distinct pull requests were approved, through 10,585 distinct (approver, pull request) acts—a pull request can be approved by more than one person—by **430 people**.

The ten busiest approvers by pull-request coverage are dims (658, 6.85%), tengqm (376), sbueringer (365), alculquicondor (335), sftim (314), liggitt (305), windsonsea (179), fabriziopandini (178), xmudrii (173) and tenzen-y (171).

Seven people are entitled to approve more than half the directories that carry a rule; 25 to 28 people are needed to cover half the approvals actually performed across the ecosystem. The two figures are different measures — a share of directories in a permission graph at a pinned commit, and a share of pull requests over eight months across 165 repositories — and the 430 approvers do not contradict the seven: 148 people hold an approval right somewhere in `kubernetes/kubernetes` alone, most over a narrow slice, and the stream spans every repository in both organizations, each with `OWNERS` files of its own. Either record alone misleads about the other: the permission files suggest a handful of people carry the project, and the approval stream would not predict that seven people are entitled to approve most of it. The next subsection puts the two on one population.

6.3 Eligible versus actual approvers, over one population

For the 1,418 merged `kubernetes/kubernetes` pull requests of Section 5.4 we report the same statistics over two relations: person was *eligible to approve* the pull request (sole-eligible), and person *did approve* it.

Eligibility over real traffic is as concentrated as the directory table suggests. liggitt was sole-eligible for **98.4%** of the 1,418 pull requests (1,396), against a 98.6% directory share: weighting the Section 6.1 directory count by observed traffic moves it by a fifth of a percentage point. Covering half the population takes **one** person by entitlement (any of the seven in Table 2 suffices alone) and **four** by acts performed — dims (23.3%), liggitt (20.0%), thockin (7.1%) and aojea together approved 744 of the 1,418 (52.5%), verified by exhaustive search, and no three people reach half. The median pull request had **11** people able to single-handedly approve it (quartiles 6 and 15; maximum 38) and 17 able to approve at least one of its files; for 56.9% of pull requests the two sets coincide, because every changed file sits under one owning scope. **42 pull requests (3.0%) had exactly one person able to approve them alone**, liggitt for 40 of them — and **three pull requests had no such person**: each touched both scheduler code and instrumentation test data, whose approver sets are disjoint, so Prow required two people jointly.

Held authority and exercised authority are now the same unit, and the gap is no longer an artifact of incommensurable measures. smarterclayton, eligible for five-sixths of the core repository's merged traffic, is recorded approving one pull request in the parsed channels over eight months; Section 10 says what that row does and does not mean about a person.

This subsection is `kubernetes/kubernetes` only, where the `OWNERS` machinery of Section 6.1 applies; the 25-to-28 figure of Section 6.2 spans both organizations and is not comparable to the four-person figure here. And checking each recorded approver against the eligibility sets also measures how much of Prow's input this corpus carries. Of the 1,604 surviving `/approve` acts on these pull requests, 94.0% came from people eligible for at least one changed file and 84.0% from people eligible for all of them. 45 pull requests (3.2%) merged with no sole-eligible recorded approver; the archive accounts for most of them: 14 had an author who was an approver for every file (the repository runs Prow with `require_self_approval: false`, so an author implicitly approves their own files), nine carried a GitHub *Approved* review with no command text, and five merged outside the window their commands sit in. Seventeen (1.2%) remain unexplained by anything in the archive. Excluding every act by a non-eligible approver moves the shares in Table 4 by at most 0.2 percentage points and swaps fourth place in the actual ranking (aojea out, wojtek-t in).

That 3.2% is the gap *within* pull requests that have at least one parsed `/approve`. The larger gap is outside them: GitHub records 1,984 `kubernetes/kubernetes` pull requests merged in the window, and 1,588 of them (80.0%) have a surviving parsed `/approve`.

Table 3: Concentration of exercised approval over the 9,601 approved pull requests, both organizations, `/approve` only. The variant column adds *Approved*-state reviews (Section 4.3), which spread the work further still; any reviewer can press that button, so the variant mixes acts Prow honored with acts it ignored.

Quantity	Commands	+ *Approved*-state
Approved pull requests	9,601	12,480
Distinct approvers	430	691
Busiest approver's coverage	658 pull requests, 6.85%	661, 5.30%
Top 1 share of acts	6.22%	4.39%
Top 5	19.35%	13.81%
Top 10	28.85%	22.56%
Top 20	40.77%	33.35%
Top 50	62.52%	53.51%
Gini over approvers	0.732	0.752
Smallest set covering half the pull requests	between 25 and 28	between 31 and 36
Greedy cover of 80%	90	114

Table 4: The seven people of Table 2 over the 1,418 merged core-repository pull requests: share for which the person was sole-eligible, and share the person is recorded approving in this corpus.

approver	eligible for	approved
liggitt	98.4%	20.0%
thockin	90.1%	7.1%
dims	86.1%	23.3%
smarterclayton	83.3%	0.1%
wojtek-t	74.4%	4.9%
dchen1107	72.9%	0.4%
deads2k	52.7%	2.9%

The other 396 (20.0%) merged with their approval carried by *Approved*-state reviews, by implicit author self-approval, or by events the archive dropped, and are invisible to the command-based figures in this paper. Section 8 records this as the paper's principal lower bound.

6.4 Leadership seats and the labeled core-repository approval stream

Every figure in this subsection is computed over SIG-labeled acts, which per Section 4.6 means the Kubernetes core repositories and not the wider ecosystem. The approver with the most pooled acts in the corpus works mostly in satellite repositories and is therefore nearly absent from these tables, which describe core-repository review.

For three SIGs, the ten busiest approvers on `/approve` alone, with roster status taken from the in-window union; each count is the number of distinct pull requests carrying that SIG's label on which the person left a surviving `/approve`. SIG Node: dims 166, liggitt 85, **SergeyKanzhelev 84 (chair)**, thockin 35, klueska 23, jsafrane 22, xmudrii 18, aojea 17, sftim 17, **dchen1107 16 (technical lead)**. SIG API Machinery: liggitt 188, dims 96, **jpbetz 88 (technical lead)**, **deads2k 67 (chair and technical lead)**, wojtek-t 66, thockin 53, **sttts 20 (technical lead)**, xmudrii 19, enj 14, aojea 12. SIG Storage: dims 61, **jsafrane 54 (technical lead)**, liggitt 51, **xing-yang 27 (chair)**, **msau42 22 (technical lead)**, xmudrii 17, thockin 16, pohly 9, sftim 8, tengqm 7.

The channel extension moved these tables in one direction: toward the roster. SIG Node's chair, SergeyKanzhelev, approves mostly through review bodies and was invisible to the issue-comment channel; with all three channels parsed they rank third in their own SIG. People who type commands as plain comments and people who approve through GitHub's review interface are different populations, and a corpus that reads only one channel systematically underrepresents the second.

Across the gated 19 SIGs, whether a SIG's busiest approver "is on the roster" depends on what counts as a seat. Each cell of Table 5 counts the SIGs whose busiest approver holds no seat under that row's definition, over the gated 19 and all 24 SIGs, ranking by `/approve` alone or pooled with `/lgtm`.

Nine distinct people top a gated SIG's `/approve` stream. Eight held a chair or technical-lead seat somewhere in Kubernetes during the window. The ninth, SIG Release's busiest approver (xmudrii, 113 pull requests), held no seat in any register we parse during the window; the July-2026 snapshot records them as a SIG technical lead, so the seat followed the work. Per Section 10 we state that per register and per window; release-team roles, for one, are recorded outside `sigs.yaml`. The distance between the registers is a matter of placement: leadership is recorded per SIG, while approval authority is recorded per path and is frequently held by someone whose seat is elsewhere.

The roster is signal, and more of it than the single-channel corpus showed. Of the 97 in-window person-seats across the gated 19, 50 (51.5%) appear in their own SIG's top ten on `/approve` and 78 (80.4%) appear somewhere in its ranking; pooled with `/lgtm`, 51 (52.6%) and 88 (90.7%). Technical leads appear more often than chairs, which is consistent with chairing being coordination work that leaves less trace in this stream.

Cross-cutting approvers, with the caveat quantified. dims ranks among the top three approvers in 16 of the 19 SIGs, and liggitt in 11. But a pull request carrying several SIG labels counts once per label, and these two are exactly the actors that flatters: their labeled approvals carry 2.27 and 3.17 labels each against a corpus mean of 1.65. Crediting each act fractionally gives dims 12 of 19 and liggitt 8. Counting only single-label acts, which discards every multi-label act rather than apportioning it and is therefore a floor rather than a credit rule, gives dims 7 and liggitt 3. As a range: **dims is a top-three approver in 12 to 16 of 19 SIGs and liggitt in 8**

Table 5: SIGs whose busiest approver holds no seat, by seat definition.

"on roster" means	19, /approve	19, pooled	24, /approve	24, pooled
chair or technical lead of that SIG, July-2026 snapshot (e210e578)	14/19	11/19	18/24	15/24
chair or technical lead of that SIG, in-window union	**11/19**	8/19	14/24	11/24
plus a Steering Committee seat or working-group chair	7/19	4/19	9/24	6/24
any formal leadership seat anywhere in the project	**1/19**	0/19	1/24	0/24

to 11, concentrated in the API-surface SIGs, depending on the credit rule, with 7 and 3 as the floors.

7 Robustness

A second attribution channel. The per-SIG results do not depend on the `sig/*` labels. Assigning each repository to a SIG through the subproject `OWNERS` URLs instead, with no labels involved, produces the same rankings where both methods observe the same pull requests (88.3% agreement over 3,518 shared pull requests; Spearman correlation 0.90 to 0.99 on testable SIGs), and it independently reproduced the off-roster finding (14 of 19 against the label channel's 12 of 19 at the time). This check ran on the single-channel corpus before the parse was extended, and it does not reach the seven SIGs whose traffic lives inside `kubernetes/kubernetes`, where repository-to-SIG mapping is undefined.

Multi-label de-fan-out. The off-roster finding is not an artifact of pull requests that carry several SIG labels and are therefore counted once per label. It holds under every credit rule: 11 of 19 under full credit, 6 under fractional, 5 under single-label. Roster recall — the share of roster seats that appear in their own SIG's top ten — *rises* when multi-label credit is removed, so if anything the fan-out was depressing that figure.

Threshold. We have no principled basis for a rank cutoff, so we treat the full curve as the result and the single cutoff as presentation. Roster recall by rank cutoff, in-window union, `/approve`: 23.7% at top-3, 38.1% at top-5, 51.5% at top-10, 60.8% at top-20, 77.3% at top-50; pooled, 25.8%, 38.1%, 52.6%, 64.9%, 81.4%. Any claim in this paper that cites a top-ten figure should be read against that curve, which rises monotonically and without a step. We note that smoothness is not itself a justification for choosing ten: it establishes only that a neighboring cutoff would tell the same story, which is the weaker property we actually rely on.

Window. dims and liggitt are top-three in 13 of the 15 combinations of test window and case SIG the archive supports (the three case SIGs of Section 6.4, across every dense window), including the window 2023-02 to 2023-10, which shares no data with ours, and their ranks in all three case SIGs are identical over the full 16.6-month dense span. This check ran on the issue-comment channel and tests the behavioral side only: we have no window-matched permission register, so nothing here says the *gap* is stable over time.

Gate. Including the five SIGs below the 50-labeled-pull-request gate moves figures in both directions: the off-roster count strengthens (14 of 24 on `/approve`) while roster recall and the cross-cutting spread weaken. The gate was not flattering the result in one direction.

Command type. Pooling `/lgtm` with `/approve` roughly doubles the ranked population (793 people issuing either command versus 460 ever issuing `/approve`) and changes SIG Node's pooled top six: bart0sh (99 pooled acts) and kannon92 are reviewers rather than approvers in the Node aliases.

Channel. Extending the parse from issue comments to all three command channels raised every behavioral count (Table 1) and moved results in one consistent direction: exercised authority spread further (cover-half 19-20 people to 25-28 across the ecosystem, 9.88% to 6.85% for the busiest share) and the roster gained recall (41.2% to 51.5% at top-10; off-roster 12 of 19 to 11 of 19), because roster-named leaders approve disproportionately through the review interface. Every single-channel figure this paper previously published moved in those directions or held; none reversed.

Identity. One login is treated as one person. A scan of all 793 approver logins surfaced two plausible duplicate pairs; merging either changes no ranking and no headline. Every login appearing in a table was checked against GitHub's account type in two AI passes; none is an automation account.

Eligibility. The Section 6.3 pipeline was adversarially reviewed before any figure was published, and three defects found by that review were fixed and the results regenerated; the regenerated output matches the reviewer's independent corrected implementation exactly. The defects and their effects, measured on the single-channel population of 1,128 pull requests the review ran against: selecting `OWNERS` snapshots without first-parent history evaluated 689 pull requests against pull-request-branch trees rather than the merged base (20 sole-eligible sets changed); comparing ISO timestamps as strings across mixed UTC offsets mis-selected snapshots for 153 pull requests (same 20 sets); and excluding vendored files, which Prow evaluates, over-credited non-dependency approvers on 78 pull requests (21 sets changed; smarterclayton 81.6% to 80.3%, deads2k 54.7% to 53.6%). No fix moved liggitt's share by more than 0.1 percentage points or changed the median, the 42 single-eligible pull requests, or the cover-half counts. Prow's directory-level approval check is weakly more permissive than per-file intersection, strictly so only when an ancestor `OWNERS` grants approvers through a non-`.*` filter, which does not occur in `kubernetes/kubernetes`; the two agree on every pull request here. Evaluating eligibility at the moment each `/approve` was typed rather than at merge time changes the recorded approver's eligibility for 4 of 1,300 acts (0.3%, measured on the issue-comment channel), although the `OWNERS` state differs between the two moments for a third of them: `OWNERS` files change often, membership of the people who actually approve rarely does. The 40 approved-but-unmerged pull requests are excluded from the population; including them would require an eligibility state that never existed.

Retraction and bots are quantified in Sections 4.3 and 4.4 and move nothing.

8 Limitations

- **One project.** Every figure describes Kubernetes. Nothing here estimates how other projects distribute approval rights, and the machinery we depend on is Kubernetes-specific.
- **The approval bit is not merge power.** Prow requires an `/lgtm` as well.
- **Directory counts are not code volume or risk.** Coverage in Section 6.1 is unweighted: a directory holding one generated file counts as much as `pkg/kubelet`. Section 6.3 answers the traffic objection — weighting by merged pull requests moves the top share from 98.6% to 98.4% — but weighting by files, churn, or risk would each give a different measure, and we report none of them.
- **Descriptive, not inferential.** The corpus is a complete parse rather than a sample, so we report point values without confidence intervals. That is appropriate for counts over a fixed corpus and inappropriate for treating any figure here as an estimate of a population parameter, including the Gini coefficient, which we report as a summary of this corpus alone.
- **One kind of authority.** Design authority exercised through enhancement proposals and design review leaves no trace in the approval stream, so a person who leads that way has few or no rows here and looks inactive. The gap runs one way: it can only make a leader look less active than they are, and the leaders it hides are disproportionately the ones the roster names, whose work is coordination rather than merge approval.
- **Scope of each result.** The per-SIG results (Section 6.4) describe the core repositories only, per Section 4.6; the eligibility results (Section 6.3) describe `kubernetes/kubernetes` only; the ecosystem figures (Section 6.2) span both organizations.
- **A 17-month archive horizon**, per Section 4.5.
- **Lower bounds.** The behavioral record parses the three command channels (Section 4.3). It still misses GitHub *Approved* reviews with no command text — 8,554 in the window, reported only as the Table 3 variant because any reviewer can press that button and only OWNERS-eligible presses count for Prow — plus implicit author self-approval and events the archive dropped. For the core repository: 20.0% of the 1,984 pull requests merged in the window carry no parsed `/approve` and are invisible to the command-based figures here, and within the 1,418 that do, 3.2% merged with no sole-eligible parsed approver (Section 6.3 decomposes them). Every count of exercised authority in this paper is a lower bound.
- **No causal or normative claim.** We do not argue that the distribution is good or bad for Kubernetes.

9 Reproducibility

The per-SIG ranking reproduces with one query against the free ClickHouse playground, with no account. We include it in Appendix A. The coverage table reproduces with a published parser, `https://github.com/jaamm/owners-coverage` (MIT), which clones the repository at a ref and prints the table; its README carries our output so a reader can distinguish a real disagreement from a flag difference. The leadership register is one public file per revision. The eligibility pipeline (file-list fetch, snapshot selection, eligibility engine, statistics, and the anomaly decomposition) is archived with its inputs and outputs, the parser at its pinned commit, the roster, and the Appendix A query at https://doi.org/10.5281/zenodo.22115990; reproducing the fetch step needs a free GitHub token, everything after it needs only the archived inputs.

Counting directories that carry an `OWNERS` file gives 501 and 98.6% for the top person. Counting only directories whose `OWNERS` file names a directory-wide approver, which sounds like the same thing, gives 413 and 98.3%, because some `OWNERS` files carry only reviewers, only narrow filters, or only emeritus entries. Both are defensible; we report the first, and the parser exposes the choice as a flag. The eligibility engine is an independent implementation of the same semantics; pointed at `7c48c2bd` it reproduces every row of Table 2, differing from the parser only in counting the repository root as a directory (502 versus 501), which changes no printed share.

10 Ethics

All data are public acts performed in a public governance process under a public protocol. We name individuals because the acts are attributable and because an unnamed claim cannot be checked, and we report aggregate counts of public acts only: no private data, no cross-domain profile, no persistent score attached to a person. Minimization was deliberate: we report counts of governance acts and coverage shares, and we do not publish adjacency between individuals, activity outside this corpus, or any composite score. We did not seek consent from the named contributors, on the grounds that the acts are public and performed under a public protocol, and we note that as a choice a reader may disagree with rather than a settled question. The foreseeable harms are that a coverage percentage can be quoted as a competence or risk judgment it does not support, and that a person's absence from a table can be read as a claim about their standing; Section 6.1 and Section 8 state the first, and the paragraphs below address the second. An organization applying this method to its own private records faces a different ethical surface, including surveillance and incentive distortion, and we would argue for aggregate-first reporting and a policy visible to the measured population.

A claim that a named person holds no formal position is more damaging, and easier to get wrong, than a claim about their activity. Formal position in this project is recorded in at least four places (SIG leadership, working groups, committees, and per-path OWNERS including subproject files), and a person absent from one of them may hold authority through another. We therefore state seat status only per register, per window, and never as a claim that someone holds no standing.

Table 4 is the most personal table in the paper, and it must be read with Section 8's lower bound: the "approved" column counts the `/approve` commands this corpus carries, and 20.0% of the core repository's merged pull requests are missing from the command record. A low figure in that column describes the recorded channel; two of the seven are listed at root as emeritus, and review conducted through design review, command-less *Approved* reviews, or outside this repository is invisible here by construction.

Disclosure. The authors work at UpTrust, a company that builds trust and authority analytics. The first author is its founder, and a companion article presenting these results appears on the company's site. That interest is why the boundary above bears restating: every figure in this paper is computed from public data with the published parser, the published pipeline, and the query in Appendix A; no proprietary tooling or non-public computation is used, and no result depends on the company's products.

11 AI assistance

This work was AI-assisted at every stage, and the specifics matter more than the label. Large language models — Anthropic Claude for analysis and drafting, with two further model lines (OpenAI Codex and xAI Grok) used for independent adversarial review — were used to: write the SQL and Python that produced every figure; draft and revise this text; search for and summarize related work; and conduct adversarial review passes whose purpose was to attack the results and the citations. The division of labor, stated plainly: the models wrote all of the analysis code and the first full draft of every section; the authors set the questions, made every framing and scoping decision, directed and adjudicated the review rounds, rewrote the abstract and Sections 1 and 2 in their own words, and edited every other section line by line, striking model phrasing where it outran the evidence. Where a sentence in this paper is wrong, a human approved it.

Verification was likewise largely model work, cross-checked between model lines, and consisted of: recomputing every database figure with deterministic ordering against the corpus described in Section 4; recomputing the coverage figures at pinned commits with the published parser and confirming the parser reproduces the printed table before release; reviewing the eligibility pipeline of Section 5.4 adversarially before any figure was published, which found three defects that were fixed and are reported in Section 7; extending the behavioral parse from one GitHub event type to three and regenerating every behavioral figure (Section 7, "Channel") — the adversarial reviews established that the archive carried channels the corpus did not and filed it as a limitation; the decision to extend the parse rather than disclose the gap was an author's, made against the models' framing; checking each reference against publisher or primary records, which found and corrected three bibliographic defects and one claim attributed to a paper that did not make it; and reading the nearest prior work in full. Figures that could not be confirmed from a primary source were removed.

That process has two limits. Model-generated summaries of external work were wrong often enough that we treat any figure not traced to a primary source as unusable, and at least one such figure survived several review passes before the primary source was read. The authors reviewed the corrections and take full responsibility for the content. Per arXiv and ACM policy, no language model is an author; this section constitutes the disclosure ACM requires.

A The per-SIG query

```
SELECT
  actor_login AS gatekeeper,
  uniqExact((repo_name, number)) AS prs_approved
FROM github_events
WHERE event_type IN ('IssueCommentEvent',
                     'PullRequestReviewEvent',
                     'PullRequestReviewCommentEvent')
 AND (repo_name LIKE 'kubernetes/%' OR repo_name LIKE 'kubernetes-sigs/%')
  AND created_at >= '2023-10-01' AND created_at < '2024-06-01'
  AND actor_login NOT LIKE '%[bot]%'
  AND actor_login NOT LIKE '%-bot'
  AND actor_login NOT LIKE '%-robot'
  AND has(labels, 'sig/node')
  AND match(body, '(?m)^/approve(\\s|$)')
  AND NOT match(body, '(?m)^/approve\\s+cancel')
GROUP BY gatekeeper
ORDER BY prs_approved DESC, gatekeeper ASC
LIMIT 10
```

Run at `https://play.clickhouse.com/play?user=play`. Swap the label for any SIG. For the pooled variant, replace the two `match` lines with the `/approve` or `/lgtm` disjunction. The secondary sort key matters more than it appears to: ties are common at the bottom of a ten-row table, and without a deterministic tie-break, two runs of the same query over the same data can return different tenth rows.

The pipeline applies stricter rules than this query (author join, self-approval removal, per-act cancel pairing), which lowers counts by roughly 5% to 12% and can swap a bottom row. The query is the reproducible artifact.

References

[1] Sabrina Aufiero, Matteo Vaccargiu, Silvia Bartolucci, Fabio Caccioli, and Giuseppe Destefanis. 2026. Coordination at Scale in Large Distributed Development: The Case of Kubernetes. In *Proceedings of the 23rd International Conference on Mining Software Repositories (MSR 2026)*. doi:10.1145/3793302.3793342

[2] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2016. A Novel Approach for Estimating Truck Factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. doi:10.1109/ICPC.2016.7503718 arXiv:1604.06766.

[3] Melvin E. Conway. 1968. How Do Committees Invent? *Datamation* 14, 4 (1968), 28–31. No DOI exists. Issue number per the scanned April 1968 issue (bitsavers.org/magazines/Datamation/196804.pdf); commonly miscited as 14(5).

[4] Kevin Crowston and James Howison. 2005. The Social Structure of Free and Open Source Software Development. *First Monday* 10, 2 (2005). doi:10.5210/fm.v10i2.1207

[5] Otávio Cury and Guilherme Avelino. 2024. Knowledge Islands: Visualizing Developers Knowledge Concentration. In *Proceedings of the XXXVIII Brazilian Symposium on Software Engineering (SBES 2024), Tools Track*. 789–795. doi:10.5753/sbes.2024.3610 arXiv:2408.08733. Title as published, without apostrophe.

[6] Georgios Gousios. 2013. The GHTorent Dataset and Tool Suite. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 233–236. doi:10.1109/MSR.2013.6624034 Title as published, including in the author's camera-ready; the project's name is GHTorrent.

[7] Elgun Jabrayilzade, Mikhail Evtikhiev, Eray Tüzün, and Vladimir Kovalenko. 2022. Bus Factor in Practice. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2022)*. doi:10.1145/3510457.3513082 arXiv:2202.01523.

[8] David Krackhardt and Jeffrey R. Hanson. 1993. Informal Networks: The Company Behind the Chart. *Harvard Business Review* 71, 4 (1993), 104–111.

[9] Jai Lal Lulla, Raula Gaikovina Kula, and Christoph Treude. 2025. Automated Code Review Assignments: An Alternative Perspective of Code Ownership on GitHub. arXiv:2512.05551 Preprint.

[10] Ehsan Mirsaeedi and Peter C. Rigby. 2020. Mitigating Turnover with Code Review Recommendation: Balancing Expertise, Workload, and Knowledge Distribution. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE 2020)*. 1183–1195. doi:10.1145/3377811.3380335

[11] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. 2016. Revisiting Code Ownership and Its Relationship with Software Quality in the Scope of Modern Code Review. In *Proceedings of the 38th International Conference on Software Engineering (ICSE 2016)*. 1039–1050. doi:10.1145/2884781.2884852

[12] Matteo Vaccargiu, Sabrina Aufiero, Silvia Bartolucci, Ronnie de Souza Santos, Roberto Tonelli, and Giuseppe Destefanis. 2026. Efficiency for Experts, Visibility for Newcomers: A Case Study of Label-Code Alignment in Kubernetes. In *Proceedings of the 30th International Conference on Evaluation and Assessment in Software Engineering (EASE 2026)*. arXiv:2603.24501. DOI not yet assigned at time of writing.