Bots Coordinating Work in Open Source Software Projects

Hukal, Philipp; Berente, Nicholas; Germonprez, Matt; Schecter, Aaron

Bots Coordinating Work in Open Source Software Projects

Philipp Hukal, Nicholas Berente, Matt Germonprez, and Aaron Schecter

DOI: https://doi.org/10.1109/MC.2018.2885970

Bots Coordinating Work in Open Source Software Projects

Philipp Hukal, Copenhagen Business School

Nicholas Berente, Mendoza College of Business – University of Notre Dame

Matt Germonprez, College of Information Science & Technology – University of Nebraska Omaha

Aaron Schecter, Terry College of Business – University of Georgia

***ABSTRACT** Bots are increasingly being used to coordinate work in open source software projects. As mechanisms that ensure the smooth functioning of open source software projects, bot activity has implications for how scholars and practitioners understand open source software development. In this article, we identify four typical classes of bots and discuss their role in the coordination of open source software projects.*

Introduction

Bots—algorithms that automatically interact with humans—are everywhere [1]. A bot could be a conversational agent on a smartphone, a participant in the chatter on social media, or a trader in the financial markets. In more and more areas non-human actors are automatically interacting with their human counterparts.

Therefore, it should come as no surprise that bots are also used to help coordinate open source software projects. From our research, we have identified four different classes of bots that help coordinate work in open source software projects: *broker*, *checker*, *gatekeeper*, and *manager* bots. In this article, we introduce these classes of bots and provide examples from Kubernetes—a popular open source software project spun out of Google. As is evident from this, and many other examples, bots are increasingly important to open source software. Recognizing these four general classes of bots is useful as they serve very different purposes. Bots support workflows of various complexities and criticalities. This gives software project organizations a lot of options in deciding where to deploy bots. Despite their widespread use in open source development, the role of bots remains unexplored in research. It is time for practitioners and scholars to recognize bots as a popular coordinating mechanism in distributed software development.

The Ubiquity and Importance of Bots

Bots are increasingly important to open source software projects. In the case of Kubernetes, the usage of bots has grown steadily. Bots are now integral to the interaction within the project community. For instance, in 2017, 97% of all source code changes in the project involved some bot activity—up from 28% during the project's first year. Over the same period, the volume and the variety of tasks involving bots have also risen substantially. Figure 1 depicts the number and the type of bot activity in the project. The chart shows a steep incline in the number (the height of each of the stacked bars) and the diversity (the colors that make up each of the stacked bars) of the various bot tasks over time. This development suggests that tasks that involve bots drive a lot of interactions in the community.

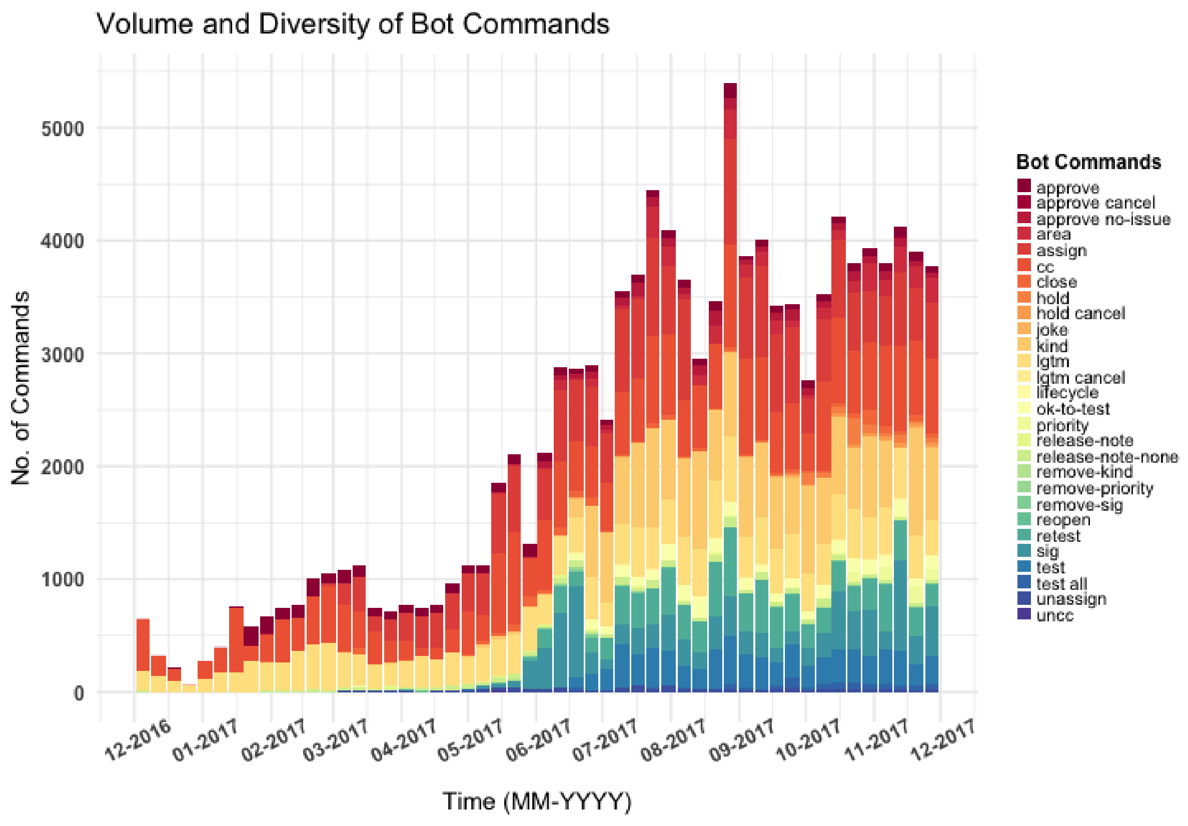


Figure 1. The volume and the diversity of bot commands used in the Kubernetes project over time. Each color in the stacked bar represents the usage of one type of bot command (chart and data licensed under Apache 2.0 by the Cloud Native Computing Foundation, Source: https://devstats.k8s.io/)

Most bot commands in Figure 1 are pre-scripted tasks that extend existing GitHub functionality to help support collaboration among developers. For instance, bot commands provide shorthand interfaces to GitHub features such as labeling discussions or patches with descriptions. However, bot activity in the project goes beyond such simple scripted commands. Bot activity routinely involves both automatic and continuous interactions with developers at crucial intersections of project workflows. Additionally, our analysis shows that bots trigger such commands themselves. Hence, bots include the above scripts and extend GitHub functionality to interact with developers.

Auditing Algorithms

We conducted an algorithm audit [2] and analyzed the activity of bots deployed in the Kubernetes source code repository on GitHub. Algorithm auditing refers to a set of analysis techniques aimed at scrutinizing—often proprietary—programmatic parts of software-enabled services. As in all algorithm audits, we seek to discover the mechanisms that determine activity of non-human actors [3].

We followed an inductive, data-driven approach to deepen our understanding of bots in open source software [4]. Kubernetes uses the online version control and management platform GitHub.com to coordinate development work [5]. One of the features of the GitHub platform is the ability of developers to collaborate on pull requests. A pull request is an activity in open source software development in which developers request to "pull" source code changes into an open source project. GitHub makes these changes, metadata, and the discussions among the developers openly accessible. We downloaded all pull request discussions in the project from the time Google open sourced the project in June 2014 to October 2017.

We include bots in our analysis based on three criteria; First, bots must be regularly active in the repository—this excludes inactive, older, or decommissioned bots. Second, the Kubernetes project organization must deploy the bots—this excludes bots under the assumed control of other developers. Third, the bots must be part of the general project workflow—this excludes bots aimed at supporting individual developers. For instance, some bots support small groups of developers whose tasks are not part of Kubernetes' overall workflow.

Bots interact with human developers mostly through comments on pull requests. Using a random sample of 150 pull requests with bot activity, we proceeded with a manual content analysis of the pull request discussions. We listed the typical actions carried out by different bots that fit the above criteria. To corroborate the activity in the pull request comments, we consulted additional information sources such as the Google groups email list used by the community and the documentation that was provided by the project. Based on the bot activity in the pull requests, we created task profiles for each bot, that is, a summary of the tasks typically performed by the bot. These profiles are the foundation for the classification of the bots which we present below.

We have identified four classes of bots in the Kubernetes project that fit the criteria above. We found that bot activity occurs from the earliest point of the project workflow—when developers first submit changes—through review, testing, and implementation of source code. Figure 2 highlights the focus of activity of four bots that are examples of the four classes of bots that we identified in the Kubernetes project.

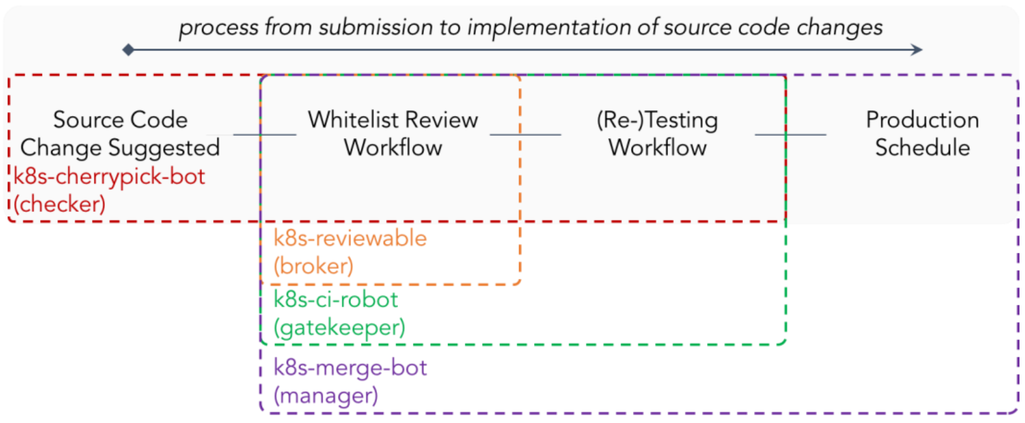

process from submission to implementation of source code changes
Source Code Change Suggested
Whitelist Review Workflow
(Re-)Testing Workflow
Production Schedule
k8s-cherrypick-bot (checker)
k8s-reviewable (broker)
k8s-ci-robot (gatekeeper)
k8s-merge-bot (manager)

Figure 2. A simplified workflow and the corresponding domains for bot activity in the Kubernetes project

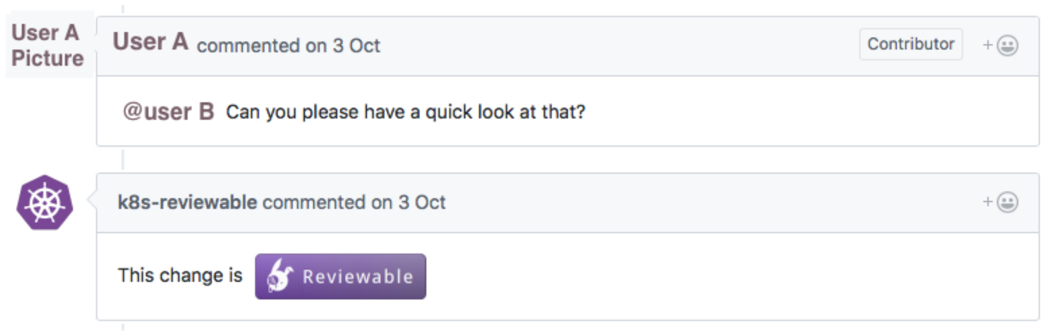
Four Classes of Bots

We identified four general classes of bots that are involved in coordinating work in the Kubernetes project. Listed in the order of the simplest to the most complicated, we refer to them as *brokers*, *checkers*, *gatekeepers*, and *managers*. Each class encompasses the functionality of the simpler classes and extends it. As a result, the complexity of bot activity increases as the bots automate higher level tasks [6]. *Brokers* scan and re-post information, whereas *checkers* additionally evaluate the information. *Gatekeepers* have formal authority associated with their evaluation and *managers* combine evaluation and formal authority with interactive coordination.

The Broker

In their simplest form, bots act as *brokers*. This class of bots follows simple *'if-this-then-that'* procedures. The objective of this class of bots is to perform brokerage-like tasks. Brokers are deployed to automate tasks that require little-to-no human involvement and tasks that do not critically impact the output of a workflow. As such, they process information (e.g., messages or comments) and relay a message to pre-defined audiences. As a simple "action support" automation [6], brokers are common in the context of social networking where their importance is increasingly recognized [1], [7]. On platforms such as GitHub, brokers are widely used, for example, to facilitate discussions among developers.

In the Kubernetes project, the bot 'k8s-reviewable-bot' is a broker. The length of the pull request discussion or the size of the suggested change in the source code triggers this bot. Above a threshold value of number of comments in the discussion or lines of code in a patch, this bot copies the communication around software changes that are currently under review. The bot automatically adds a comment to the respective pull request discussion with a link to an external application. The link directs the developers to a tool that the project organization prefers to use to keep track of complex patches. In the following example, the bot refers to the location where the suggested changes in the source code are tracked, grouped, and visualized for review. Figure 3 illustrates how the bot is making the project members aware of the option to follow the review process elsewhere. This bot completes this task by posting a permanent link to the external review environment ('reviewable') in the pull request discussion.


User A Picture

User A commented on 3 Oct

Contributor

@user B Can you please have a quick look at that?

k8s-reviewable commented on 3 Oct

This change is Reviewable

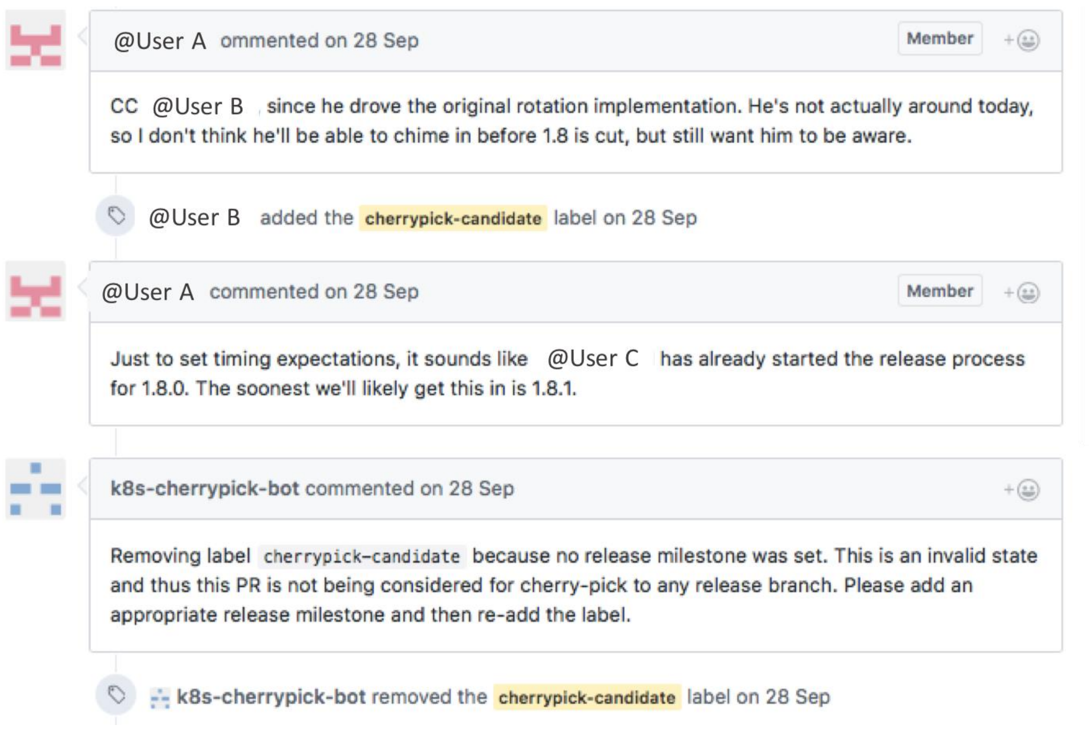
Figure 3. A Kubernetes broker-bot automatically posting to a pull request discussion ('k8s-reviewable-bot')

Source: Kubernetes Code Repository on GitHub.com

The Checker

Another class of bots found in open source software projects is the *checker*. Checkers follow simple rules similar to brokers. A crucial difference is that checkers additionally includes an intervention to automate decision making support [6]. Checkers are triggered when information is scanned according to the pre-defined rules. Similar to the process of using a dictionary, checkers use a look-up and cross-reference process for the information that is presented using the pre-set rules. Thus, checkers can ensure consistency within the area that they monitor. Aside from this difference in their objective, checkers also have a different scope than brokers. Checkers regularly involve human actors by alerting them of the necessity to double-check information or communicating when an intervention is needed. Consequently, it is possible to deploy checkers in support of workflows that are more critical to the final development output.

In the Kubernetes project, the bot 'k8s-cherrypick-robot' is a checker. The bot leverages a core feature on the GitHub platform to help developers coordinate their work. GitHub assigns free-text labels to each conversation. Standard labels include tagging a conversation as a bug, an enhancement, or a question. GitHub also allows to create custom labels. One such custom label in the Kubernetes project is the ‘cherrypick’. So-called ‘cherrypicks’ are patches meant for major releases of the project. As such, changes with that label are typically passed on to whitelisted contributors for further review. The bot is triggered when a developer marks a source code change with the ‘cherrypick’-label to help to support this workflow. By pre-screening the ‘cherrypick’ labels and assigning or removing descriptions, the checker acts as a mediator between the wider developer community and whitelisted project members. In Figure 4, community members discuss the inclusion of a suggested change in the project’s release cycle and propose to label their patch as a ‘cherrypick’. However, the intended major release has already been closed and no longer accepts submissions. The k8s-cherrypick-bot intervenes as a consequence of the patch not being associated with a major release and removes the ‘cherrypick’-label automatically.



@User A ommented on 28 Sep

CC @User B since he drove the original rotation implementation. He's not actually around today, so I don't think he'll be able to chime in before 1.8 is cut, but still want him to be aware.

@User B added the cherrypick-candidate label on 28 Sep

@User A commented on 28 Sep

Just to set timing expectations, it sounds like @User C has already started the release process for 1.8.0. The soonest we'll likely get this in is 1.8.1.

k8s-cherrypick-bot commented on 28 Sep

Removing label `cherrypick-candidate` because no release milestone was set. This is an invalid state and thus this PR is not being considered for cherry-pick to any release branch. Please add an appropriate release milestone and then re-add the label.

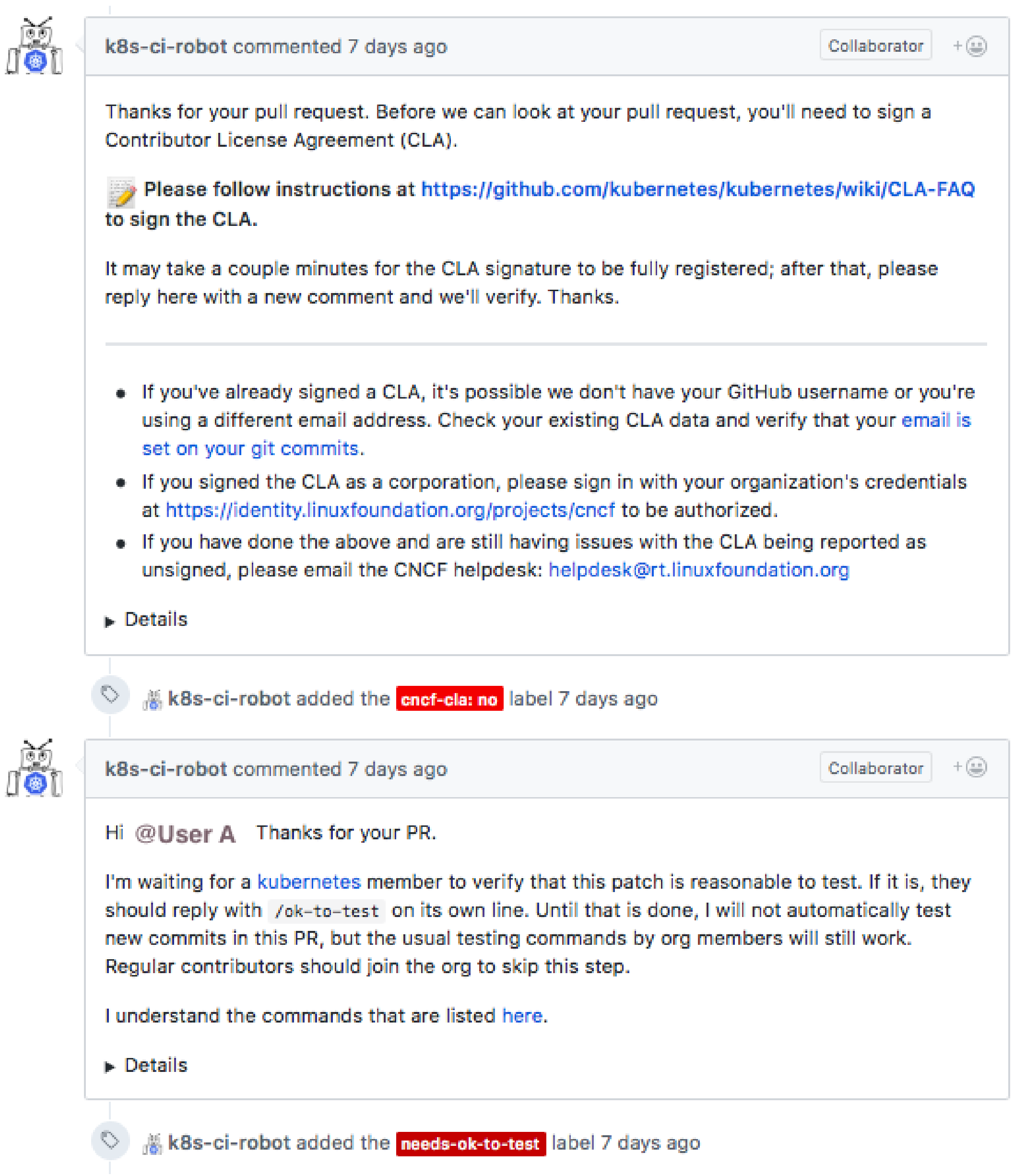
k8s-cherrypick-bot removed the cherrypick-candidate label on 28 Sep

Figure 4. A Kubernetes checker-bot commenting (‘k8s-cherrypick-robot’) [Note: the conversation was truncated for illustration] Source: Kubernetes Code Repository on GitHub.com

The Gatekeeper

Gatekeepers automatically exercise control [6] over parts of the workflow. Therefore, this bot class supports actions with higher levels of criticality for the development of open source software. When deployed at crucial intersections of the project workflow, gatekeepers confirm adherence to predefined requirements so that work can proceed to subsequent steps in the process. Bots of this class differ from the previous classes mentioned above because their activities exercise direct control over the workflow. Activity in the preceding bots could have been ignored without jeopardizing the project workflow. In contrast, the gatekeeper bot enforces a mandatory checkpoint for developer activity at a crucial bottleneck. Without passing the gatekeeper bot, work will not proceed.

In the Kubernetes project, the bot 'k8s-ci-bot' is a gatekeeper bot. Deployed in the general testing workflow, the bot first checks whether the author of a suggested change in the source code has signed the Contributor Licensing Agreement (CLA). If necessary, the bot then redirects to the respective compliance page before the developer can submit changes for review (Figure 5).



k8s-ci-robot commented 7 days ago

Collaborator

Thanks for your pull request. Before we can look at your pull request, you'll need to sign a Contributor License Agreement (CLA).

Please follow instructions at https://github.com/kubernetes/kubernetes/wiki/CLA-FAQ to sign the CLA.

It may take a couple minutes for the CLA signature to be fully registered; after that, please reply here with a new comment and we'll verify. Thanks.

- If you've already signed a CLA, it's possible we don't have your GitHub username or you're using a different email address. Check your existing CLA data and verify that your email is set on your git commits.
- If you signed the CLA as a corporation, please sign in with your organization's credentials at https://identity.linuxfoundation.org/projects/cncf to be authorized.
- If you have done the above and are still having issues with the CLA being reported as unsigned, please email the CNCF helpdesk: helpdesk@rt.linuxfoundation.org

▶ Details

k8s-ci-robot added the cncf-cla: no label 7 days ago

k8s-ci-robot commented 7 days ago

Collaborator

Hi **@User A** Thanks for your PR.

I'm waiting for a kubernetes member to verify that this patch is reasonable to test. If it is, they should reply with `/ok-to-test` on its own line. Until that is done, I will not automatically test new commits in this PR, but the usual testing commands by org members will still work. Regular contributors should join the org to skip this step.

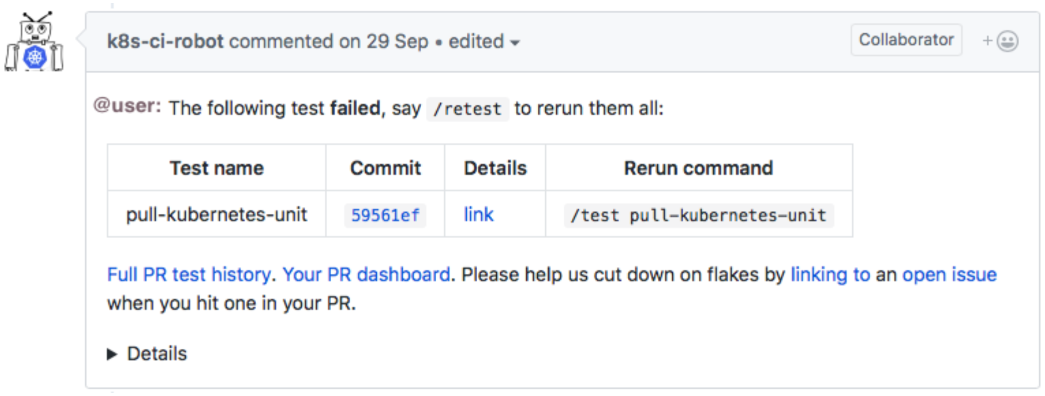
I understand the commands that are listed here.

▶ Details

k8s-ci-robot added the needs-ok-to-test label 7 days ago

Figure 5. A Kubernetes gatekeeper bot screening a pull request for CLA compliance ('k8s-ci-robot')
Source: Kubernetes Code Repository on GitHub.com

If CLA compliance is confirmed, the bot checks if the change in the source code has been properly labeled and reviewed according to the procedures agreed to by the project organization. Only once all preceding checks are met does the gatekeeper automatically forward the source code change to standardized testing. Figure 6 illustrates how the bot communicates the code test results and coordinates further steps.


k8s-ci-robot commented on 29 Sep • edited ▾ Collaborator

@user: The following test **failed**, say `/retest` to rerun them all:

Test name	Commit	Details	Rerun command
pull-kubernetes-unit	`59561ef`	link	`/test pull-kubernetes-unit`

Full PR test history. Your PR dashboard. Please help us cut down on flakes by linking to an open issue when you hit one in your PR.

▶ Details

Figure 6. A Kubernetes gatekeeper bot sharing test results (‘k8s-ci-robot’)
Source: Kubernetes Code Repository on GitHub.com

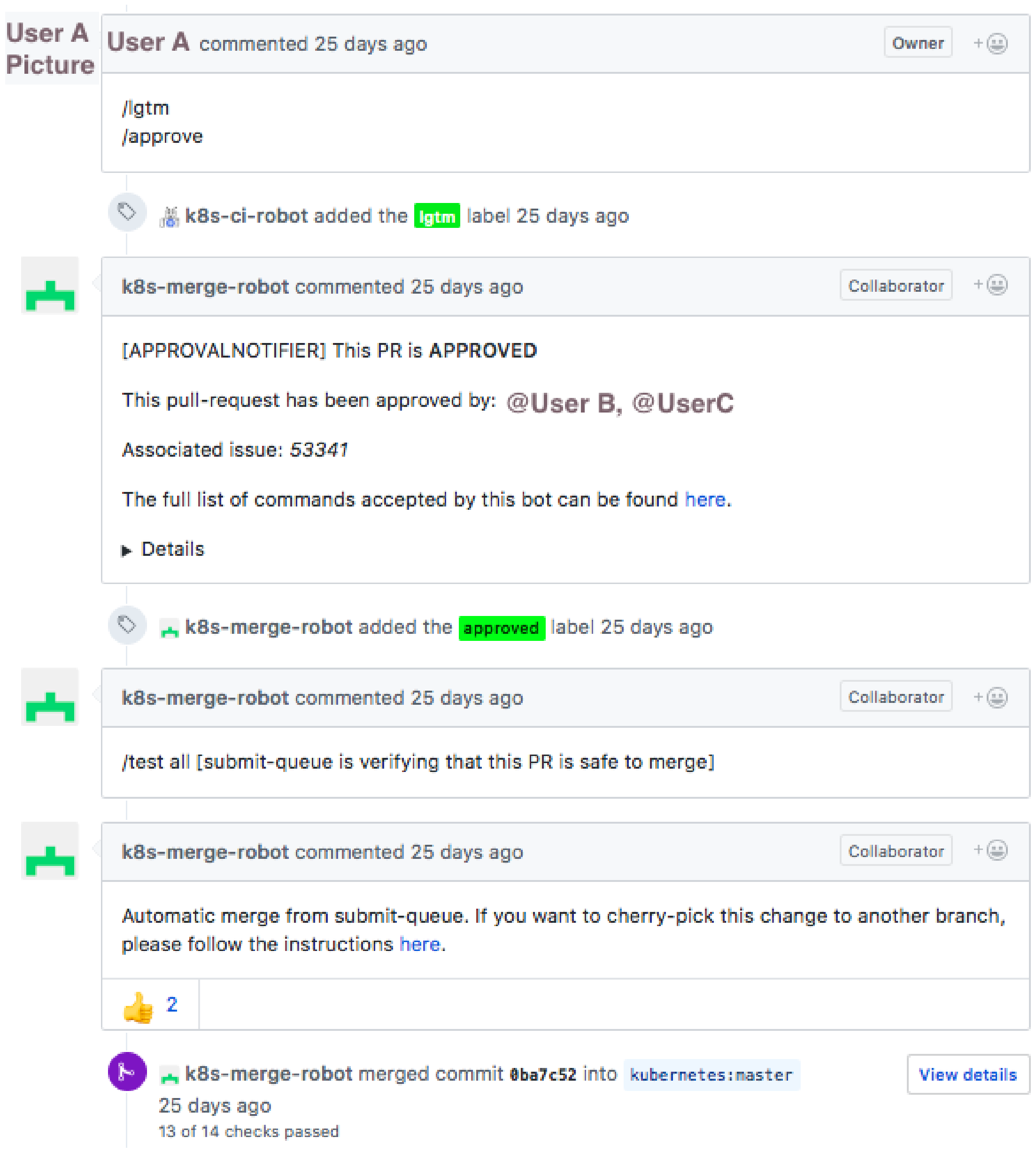
<u>The Manager</u>

Managers are deployed to complex and critical tasks in open source software development. While this class includes functions that are also carried out by other bots, their activity extends to vital development tasks—often hand-in-hand with developers. One unique functionality of managers is the allocation and prioritization of resources to their workflow. Thus, managers help guide work by automating supervisory control [6].

In the Kubernetes project, the bot ‘k8s-merge-bot’ is a manager. The bot is triggered when a source code change has been submitted by checking if the developer is on a whitelist. If not, the bot assigns reviewers from the whitelist and tasks them with a review of the proposed changes. After review by a whitelisted developer of the project, the manager feeds the patch into the testing workflow monitored by a gatekeeper (i.e., k8s-ci-robot). Upon completion of the standardized tests, the manager schedules the changes for production according to a prioritization that is defined by the project developers. This prioritization occurs by confirming that no dependencies or redundancies occur during the time that is needed to review and test a patch. Only the k8s-merge-bot and a limited number of project developers have permission to schedule source code changes for implementation.

Additionally, the k8s-merge-bot undertakes a wide spectrum of actions. For instance, triggered by checking timestamps on patches, the bot reminds project developers that a discussion has not been active for a while. Additionally, the bot plugs patches into a variety of starting points across project workflows. For example, the bot controls the re-testing workflow by flagging patches with so-called ‘flaky’ (i.e., inconsistent) test results. When a patch has been identified as ‘flaky’ the manager re-directs the ‘flaky’ patch back to the testing workflow. This process repeats itself until the test results are no longer suspicious. The bot also ensures that free-text descriptions by users (such as labels) are applied consistently to support developer coordination.

In Figure 7, the ‘k8s-merge-bot’ is triggered by a reviewing developer through the command “/approve”. The bot confirms approval of whitelisted developers, automatically re-tests the patch, and finally schedules the source code change for production. It does so by automatically calling the “/test” command which triggers another bot and initiates the testing workflow.


User A Picture

User A commented 25 days ago · Owner

/lgtm
/approve

k8s-ci-robot added the lgtm label 25 days ago

k8s-merge-robot commented 25 days ago · Collaborator

[APPROVALNOTIFIER] This PR is **APPROVED**

This pull-request has been approved by: @User B, @UserC

Associated issue: *53341*

The full list of commands accepted by this bot can be found here.

▶ Details

k8s-merge-robot added the approved label 25 days ago

k8s-merge-robot commented 25 days ago · Collaborator

/test all [submit-queue is verifying that this PR is safe to merge]

k8s-merge-robot commented 25 days ago · Collaborator

Automatic merge from submit-queue. If you want to cherry-pick this change to another branch, please follow the instructions here.

👍 2

k8s-merge-robot merged commit `0ba7c52` into `kubernetes:master` 25 days ago

13 of 14 checks passed

Figure 7. A Kubernetes manager-bot coordinating work to completion ('k8s-merge-robot')
Source: Kubernetes Code Repository on GitHub.com

Open Source Bots and Organizations

Open source software development communities were long thought to resemble egalitarian bazaars [8]. In these communities, individuals found interesting ways to democratically coordinate their work, resulting in surprisingly complex code [9]–[11]. This phenomenon is interesting because it is difficult to coordinate complex activities without the benefit of an organizational hierarchy such as those found in corporations. Increasingly, researchers and practitioners have come to realize that open source software projects are not as egalitarian as once perceived. Large-scale open source software projects are structured and regularized in ways that connect with organizational work. As such, open source software projects can be quite hierarchical [12] and often involve a significant amount of corporate engagement [13], [14].

In these corporate contexts, bots are one of many instruments in the coordination and structuring of open source software project work. While bots are by no means an attribute exclusively found in open source software projects with corporate engagement, it is important to recognize that the deployment of bots makes sense for corporations – as sponsors as well as users of open source software projects. In the context of a maturing open source software development, bots are one of many indicators of an increasing move towards professional development practices in open source projects.

Our illustrations, using data from Kubernetes, suggest that bots provide procedural control to maintain and reinforce order in the project. First, with the help of bots, developers extend their ability to coordinate development in-line with the agreed-upon procedures. Second, bots take over mundane tasks (e.g., the checker bot) which in turn frees up developer efforts so that they can focus their energy elsewhere. Finally, bots accurately relay feedback. For example, the checker bots and the manager bots are unequivocal about failed tests and articulate exactly what is expected from developers. Enforcing the agreed-upon rules for quality and process facilitates collaborative work in the project. Collectively, these factors provide stability and reliability at scale.

This is a critical aspect for the continued use of the software. As projects grow in size, bots represent an important mechanism for reinforcing and managing the procedures that were initially agreed-upon to coordinate the development work for a project. Bots increase the reliability and stability that corporations require to build on and trust open source software. It is no coincidence that Kubernetes was once an internal Google project and now involves a variety of corporations—some of its biggest contributors are Red Hat, Microsoft, and Huawei. Corporations likely favor open source projects with a certain level of domestication to be useful [15]. Nobel laureate Herb Simon argued that it is *procedural rationality* which enables the execution of complex tasks in organizations [16]. By addressing the housekeeping, quality control, and routing of common activities, bots help to keep work in open source projects predictable and enforce procedural rationality. Bots augment developer reach and enforce the rules of the project by enacting desired procedures through the execution of various tasks—often automatically. Akin to clerks, bots instantiate process intentions, making them important stabilizing mechanisms for open source software projects.

Conclusion

How important bots are to open source projects is highlighted by the four general classes of bots that we identified and illustrated. As illustrated by our case study, bots leverage and extend the functionality of version control repository management systems such as GitHub. As a popular representative of such systems, GitHub provides ample sources of projects using bots in their development work. For instance, we found evidence for bots similar to our classification in a variety of prominent open source projects such as Node.js (https://github.com/nodejs), Bootstrap (https://github.com/twbs), and brew (https://github.com/Homebrew). Indeed, Google's internal software development operation relies substantially on bots such as the ones described here [17]. For our research, we fit bots into one of four classes. Further insights might come from investigations that were beyond our focus and might discover classes of bots that we did not address in this paper. Consider, for instance, that some projects use bots to automate singular yet cumbersome tasks such as code migration. Additionally, certain bots operate across projects because of an attribute these projects share, for example, their programming language. There are all sorts of bots that can conceivably impact open source projects – from code generating bots to malicious bots. Our research offers a view on bots that help with routine coordination within one project. This highlights the importance of bots in open source software projects and proposes a parsimonious taxonomy of bots that help coordinating work on those projects.

The role of bots has implications for how practitioners and scholars think about open source software projects. Bots serve as coordination mechanisms that are important to the smooth functioning of open source software projects. For practitioners, bots are a useful tool for implementing and understanding project workflows. Software development practices can be supported and advanced through bot deployment by covering various degrees of task complexities and criticalities. For scholars, the presence of bots affects ways of researching open source software projects, especially in applications of social network analysis or sequence analyses. It is critical to incorporate an understanding of bots when attempting to explore the dynamics of open source software project relationships and processes. Research on open source software projects that does not actively attend to the role of bots risks conflating different forms and purposes of activity in a project. Disregarding bot activity as uninteresting altogether may restrict the conclusions that can be drawn from studies on open source software projects.

Lastly, it is important to understand bots in the context of the maturing open source software space. Of course, such projects still involve hackers creating software for their own needs, but open source software is increasingly part of many commercial operations [18]. As bots extend developer influence, they help to enforce procedural rules to implement pre-defined workflows; their existence indicates much more regularized work than previously discussed in studies of open source [19].

Acknowledgement

This project received funding from the Alfred P. Sloan Foundation Digital Technology grant on Open Source Health and Sustainability, Num: 8434 (https://sloan.org/grant-detail/8434).

About the Authors

Philipp Hukal is an Assistant Professor at Copenhagen Business School, Department of Digitalization. His research examines digitally-enabled innovation within and across organizations covering topics such as digital platforms, open source software development, and digital entrepreneurship. He holds a PhD in Information Systems and Management from Warwick Business School, University of Warwick (UK).

Nicholas Berente is an Associate Professor in the Mendoza College of Business, University of Notre Dame. He studies how digital innovation drives large-scale change in organizations and institutions. He teaches courses on IT Strategy and Digital Innovation. Prof. Berente received his PhD from Case Western Reserve University's Weatherhead School of Management. He was an entrepreneur prior to his academic career, founding two technology companies. He is the principal investigator for a number of U.S. National Science Foundation projects and has won multiple awards for his teaching and his research. Prof. Berente is associate editor for MIS Quarterly.

Matt Germonprez is the Mutual of Omaha Associate Professor of Information Systems in the College of Information Science & Technology, University of Nebraska at Omaha. He uses qualitative field-studies to research corporate engagement with open communities and the dynamics of design in these engagements. His lines of research have been funded by numerous organizations including the National Science Foundation, the Alfred P. Sloan Foundation, and Mozilla. Matt is the co-founder of the Association for Information Systems SIGOPEN and the Linux Foundation Community Health Analytics OSS Project (CHAOSS).

Aaron Schecter is an Assistant Professor of Management Information Systems at the University of Georgia, Terry College of Business. His research focuses on dynamic theories of team functioning and the development of supporting analytical methods. He holds a PhD from Kellogg School of Management, Northwestern Universtiy.

References

[1] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini, "The Rise of Social Bots," *Commun. ACM*, vol. 59, no. 7, pp. 96–104, 2016.

[2] N. Diakopoulos, "Accountability in Algorithmic Decision Making," *Commun. ACM*, vol. 59, no. 2, pp. 56–62, 2016.

[3] S. Wachter, B. Mittelstadt, and L. Floridi, "Transparent, Explainable, and Accountable AI for Robotics," *Sci. Robot.*, vol. 2, no. 6, pp. 1–17, 2017.

[4] N. Berente, S. Seidel, and H. Safadi, "Data-Driven Computationally-Intensive Theory Development," *Inf. Syst. Res.*

[5] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social Coding in GitHub : Transparency and Collaboration in an Open Software Repository," in *CSCW Feburary 11-15*, 2012.

[6] M.-P. Pacaux, S. Debernard, A. Godin, B. Rajaonah, F. Anceaux, and F. Vanderhaegen, “Levels of Automation and Human-Machine Cooperation: Application to Human-Robot Interaction,” in *Proceedings of the 18th World Congress of The International Federation of Automatic Control Aug 28 - Sep 2*, 2011.

[7] C. Salge and E. Karahanna, “Protesting Corruption on Twitter: Is it a bot or is it a person,” *Acad. Manag. Discov.*, 2016.

[8] E. Raymond, “The Cathedral and the Bazaar,” *Philos. Technol.*, vol. 12, no. 3, 1999.

[9] A. Lindberg, N. Berente, J. Gaskin, and K. Lyytinen, “Coordinating Interdependencies in Online Communities: A Study of an Open Source Software Project,” *Inf. Syst. Res.*, vol. 27, no. 4, pp. 751–772, 2016.

[10] J. Howison and K. Crowston, “Collaboration Through Open Superstition: A Theory of The Open Source Way,” *MIS Q.*, vol. 38, no. 1, pp. 29–50, 2014.

[11] A. Aaltonen and G. F. Lanzara, “Building Governance Capability in Online Social Production: Insights from Wikipedia,” *Organ. Stud.*, vol. 36, no. 12, pp. 1649–1673, 2015.

[12] T. Cornford, M. Shaikh, and C. Ciborra, “Hierarchy, Laboratory and Collective: Unveiling Linux as Innovation, Machination and Constitution,” *J. Assoc. Inf. Syst.*, vol. 11, no. 12, pp. 809–837, 2010.

[13] M. Germonprez, J. P. Allen, B. Warner, J. Hill, and G. McClements, “Open Source Communities of Competitors,” *Interactions*, vol. 20, no. 6, pp. 54–59, 2013.

[14] M. Germonprez, J. E. Kendall, K. E. Kendall, L. Mathiassen, and B. Young, “A Theory of Responsive Design: A Field Study of Corporate Engagement with Open Source Communities,” *Inf. Syst. Res.*, vol. 28, no. 1, pp. 64–83, 2017.

[15] C. Kelty, “There Is No Free Software,” *J. Peer Prod.*, 2013.

[16] H. Simon, *Sciences of the Artificial*. MIT Press, 1996.

[17] R. Potvin and J. Levenberg, “Why Google stores billions of lines of code in a single repository,” *Commun. ACM*, vol. 59, no. 7, pp. 78–87, 2016.

[18] B. Fitzgerald, “The Transformation of Open Source Software,” *MIS Q.*, vol. 30, no. 3, pp. 587–598, 2006.

[19] B. Butler, E. Joyce, and J. Pike, “Don’t look now, but we’ve created a bureaucracy: the nature and roles of policies and rules in Wikipedia,” in *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, 2008.