

DevOps

More than just YAML

Holland Gibson

ADMIN

Billable to:

- FED portfolio (under Michelle H.)
- COMM/INTL portfolio (under Joel M.)
- PROFDEV

Who am I?

Who am I?

Optix-Edge Fed

Who am I?

Optix-Edge Fed

@ghgibson

DevOps

DevOps

IT but with a different title?

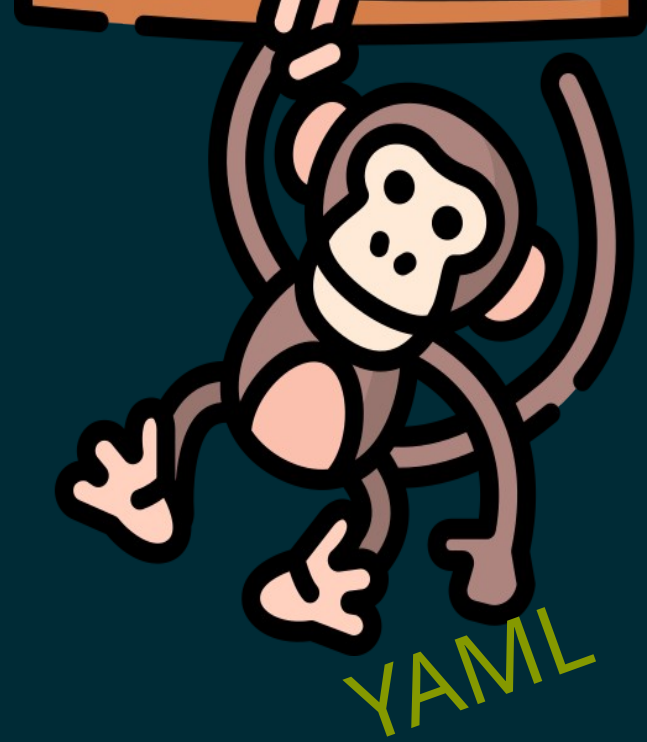
DevOps

IT but with a different title?

Kubernetes (k8s)?

DevOps

IT but with a different title?



Kubernetes (k8s)?

Ask a programmer to review
ten lines of code, [they'll] find
ten issues.

Ask a programmer to review ten lines of code, [they'll] find ten issues. Ask [them] to do five hundred lines, and [they] will say it looks good.

Ask a programmer to review ten lines of code, [they'll] find ten issues. Ask [them] to do five hundred lines, and [they] will say it looks good.

Gene Kim, "The DevOps Handbook"

Solving this problem:

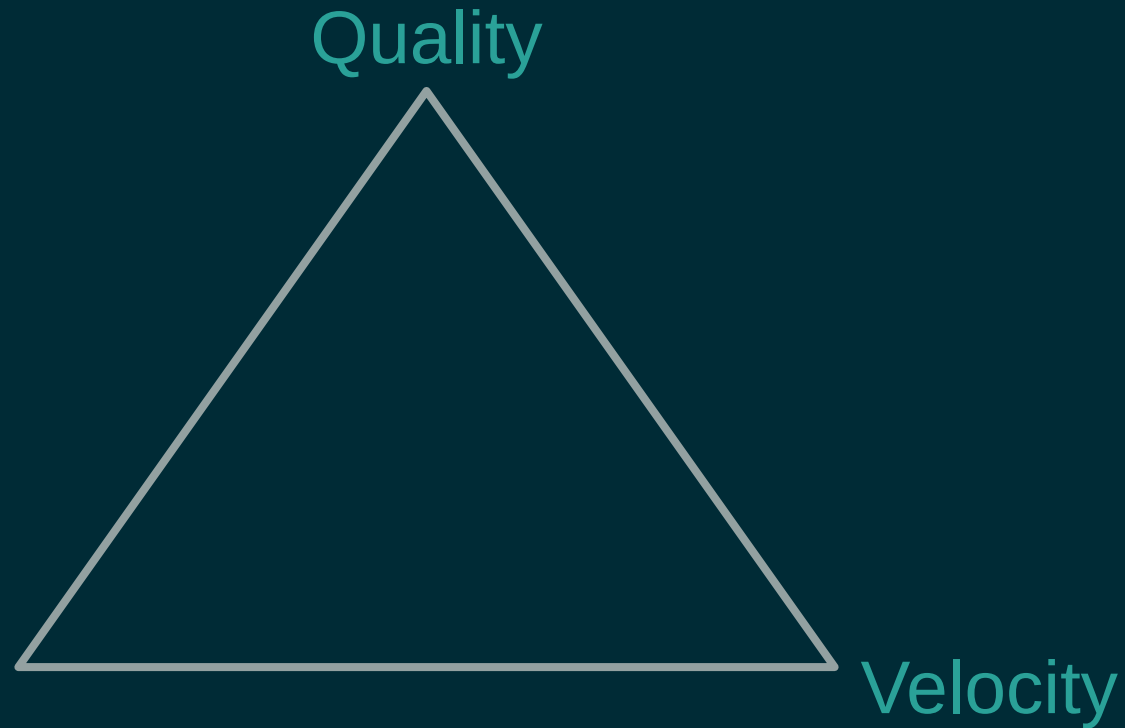


Solving this problem:

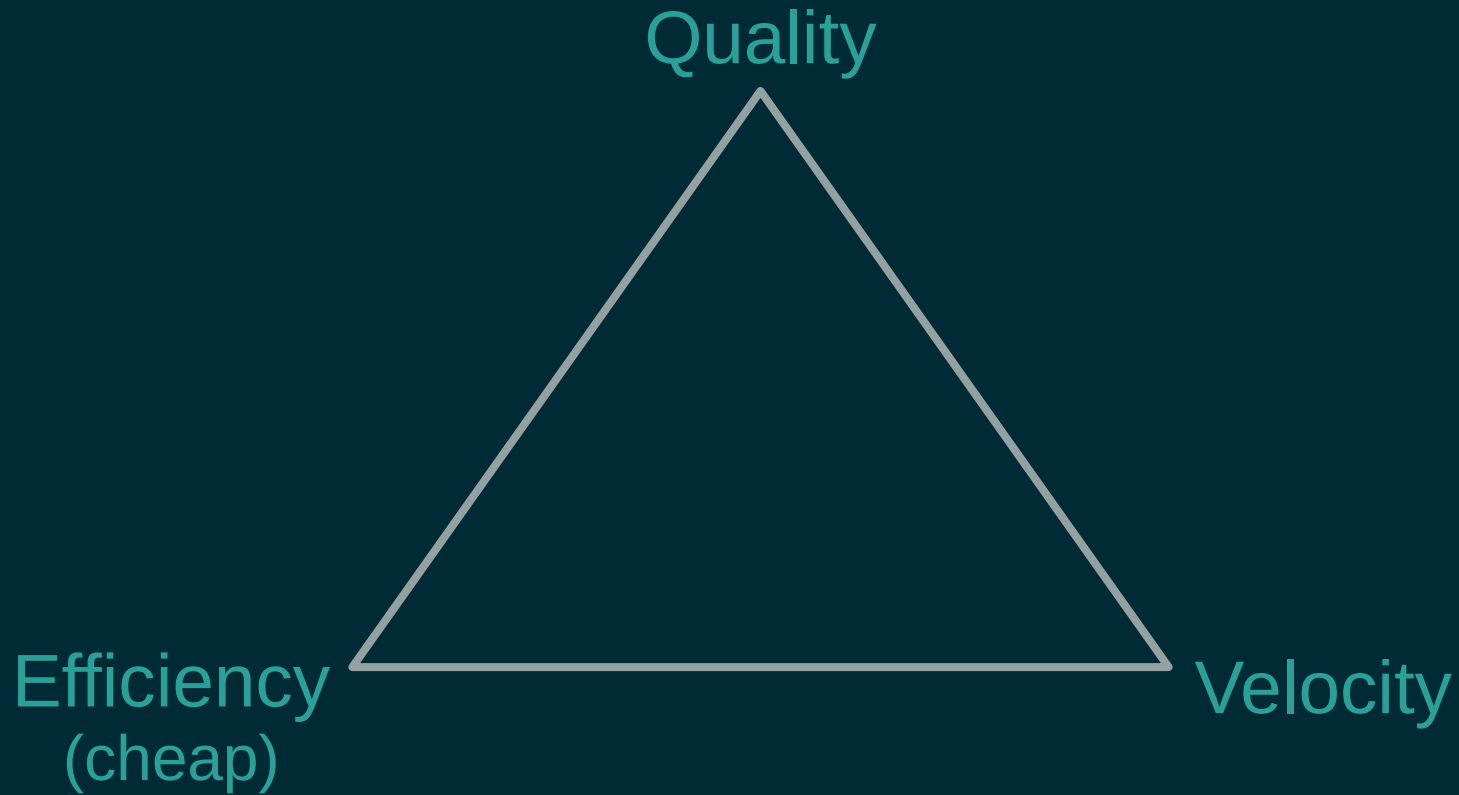
Quality



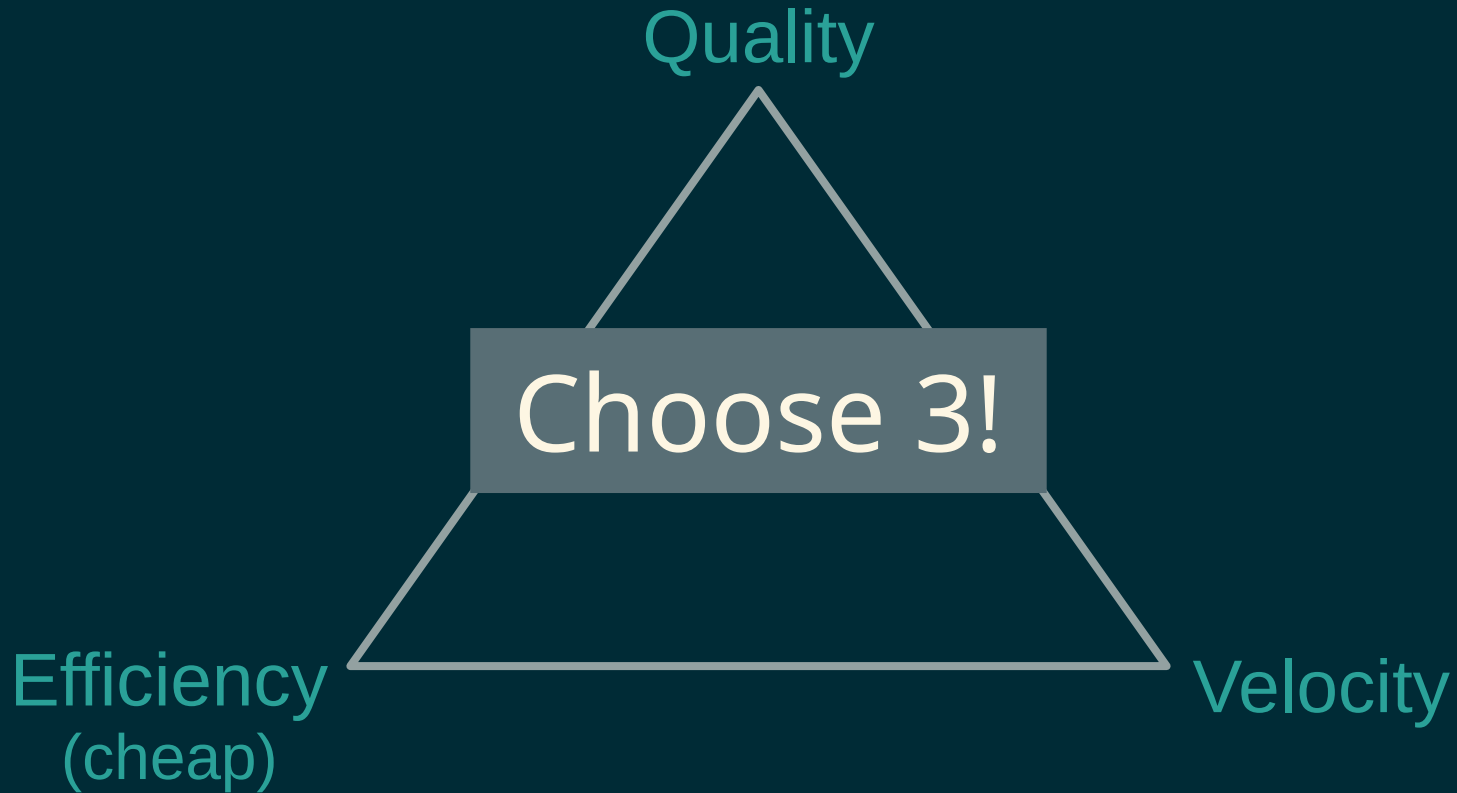
Solving this problem:



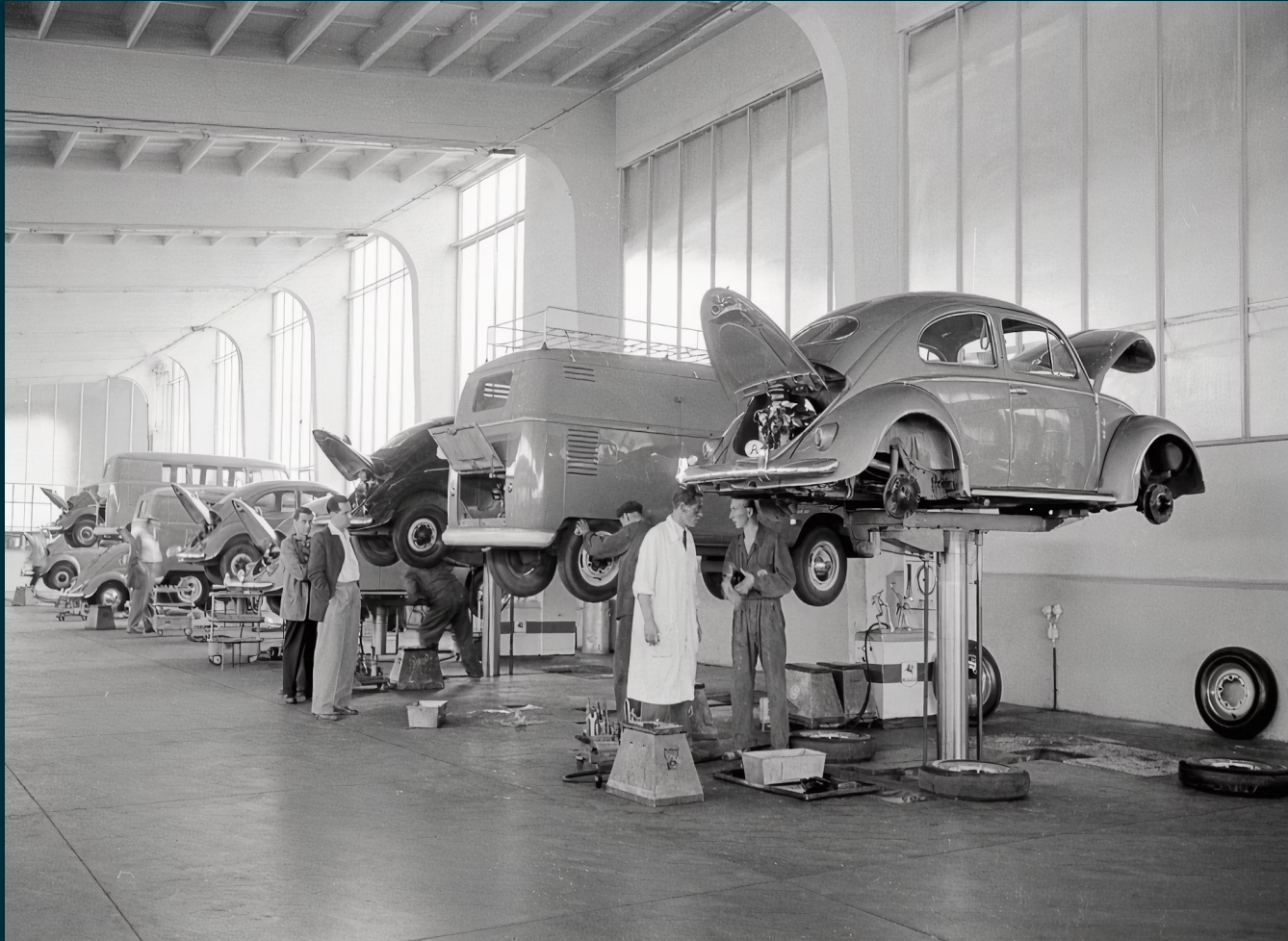
Solving this problem:



Solving this problem:



Developers in Operations



How to DevOps



Elite



Low

DORA '23 takeaways



DORA '23 takeaways



- culture

DORA '23 takeaways



- culture
- users in mind

DORA '23 takeaways



- culture
- users in mind
- faster code reviews

DORA '23 takeaways



- culture
- users in mind
- faster code reviews
- quality documentation → amplifies tech. ability

DORA '23 takeaways



- culture
- users in mind
- faster code reviews
- quality documentation → amplifies tech. ability
- infrastructure flexibility (cloud)

DORA '23 takeaways



- culture
- users in mind
- faster code reviews
- quality documentation → amplifies tech. ability
- infrastructure flexibility (cloud)
- Balance delivery speed, operational performance, and user focus

DORA '23 takeaways



- culture
- users in mind
- faster code reviews
- quality documentation → amplifies tech. ability
- infrastructure flexibility (cloud)
- Balance delivery speed, operational performance, and user focus
- Distribute work fairly

DORA '23 takeaways



- culture
- users in mind
- **faster code reviews**
- quality documentation → amplifies tech. ability
- infrastructure flexibility (cloud)
- Balance delivery speed, operational performance, and user focus
- Distribute work fairly

Review Speed

Review Speed

- 1 day review – Google

Review Speed

- 1 day review – Google
- 10 min build – Kent Beck “Extreme Programming Explained”

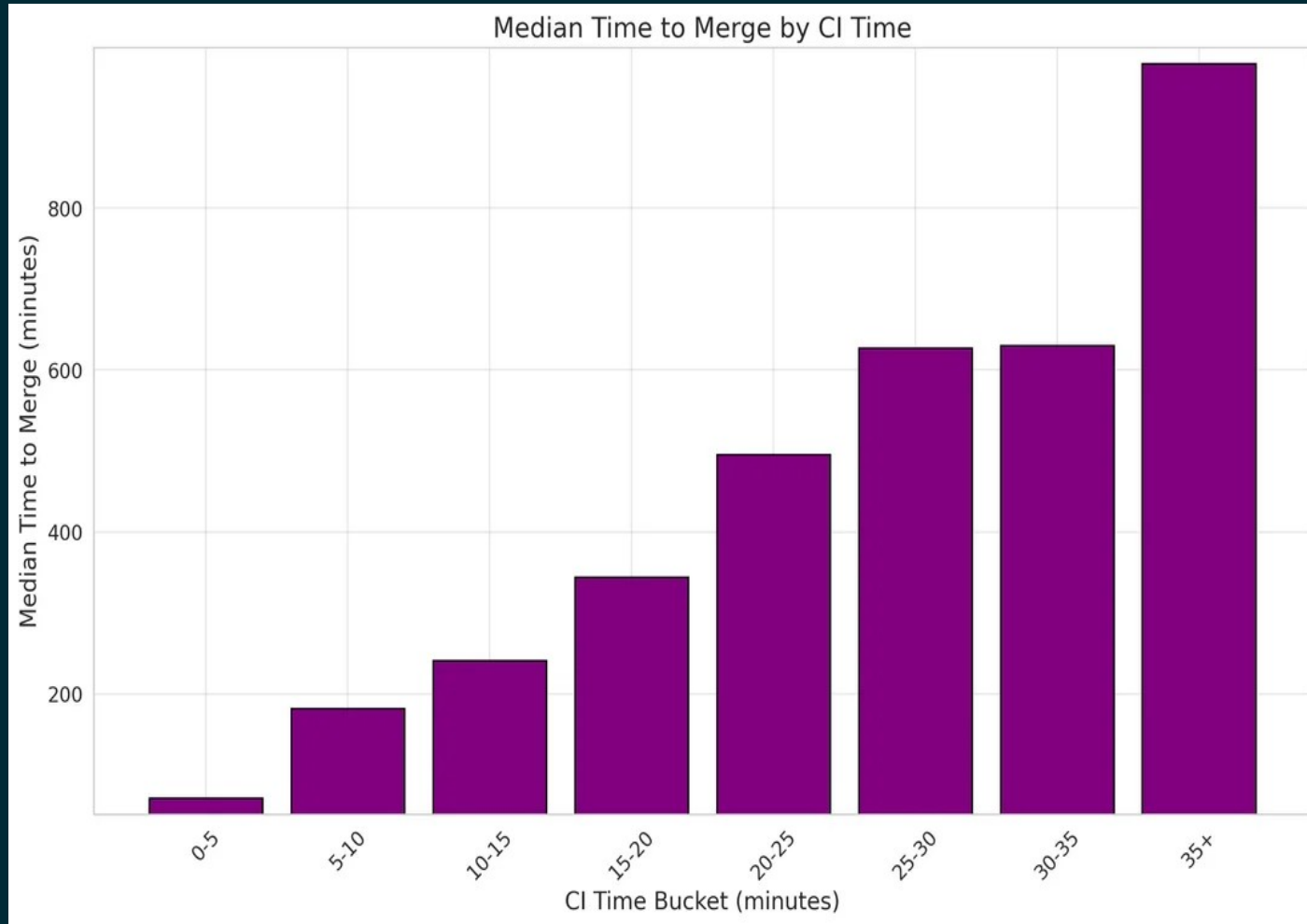
Review Speed

- 1 day review – Google
- 10 min build – Kent Beck “Extreme Programming Explained”



“How long should your CI take?”





Our Numbers

REDACTED

CI Speed



CI Speed

- No lower bound

CI Speed

- No lower bound
- 1-2 hour merge penalty / for every +5 min CI

CI Speed

- No lower bound
- 1-2 hour merge penalty / for every +5 min CI

`graphite.dev` - “How long should your CI take?”

Side-adventure: Our stats

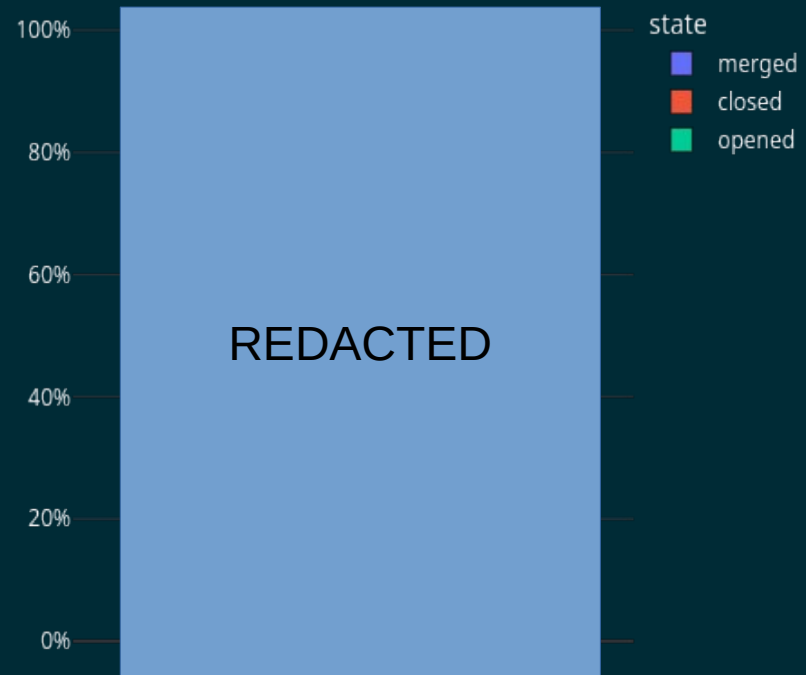
Side-adventure: Our stats



Side-adventure: Our stats

*Only visible projects with > 25 MRs

MR Status

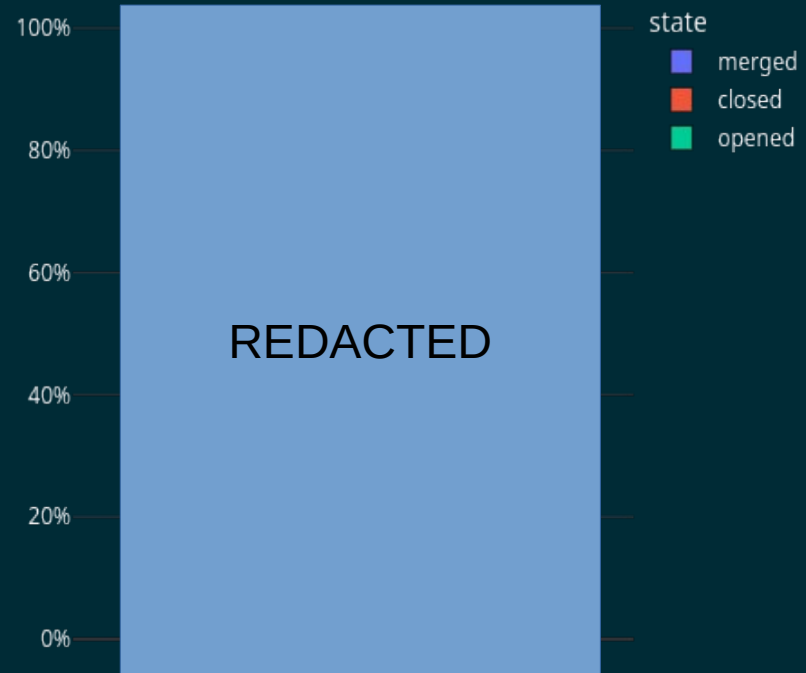


Side-adventure: Our stats

*Only visible projects with > 25 MRs

> 23k MRs

MR Status



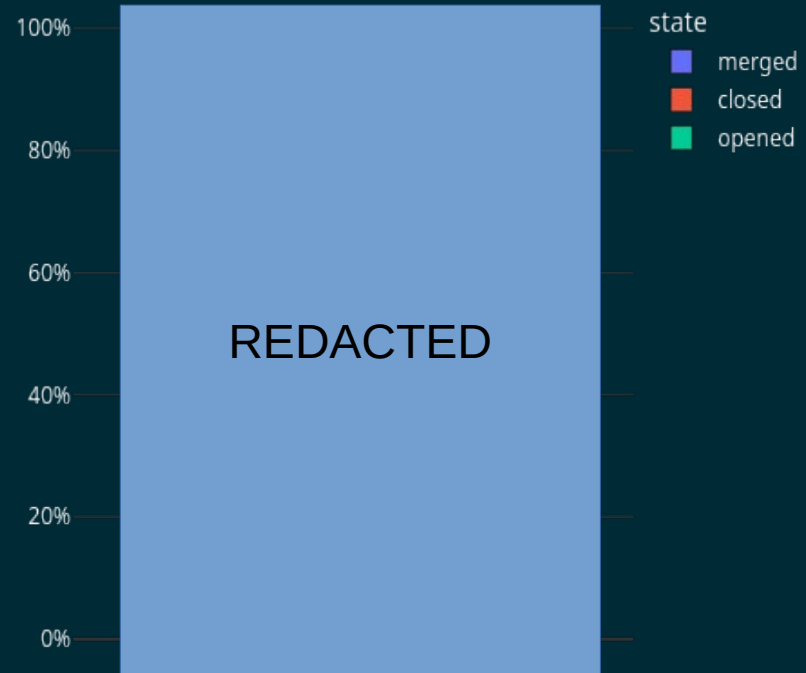
Side-adventure: Our stats

*Only visible projects with > 25 MRs

> 23k MRs

[REDACTED PROJECT] > 6k

MR Status



CI Jobs

- 58% of MRs: 1+ pipeline

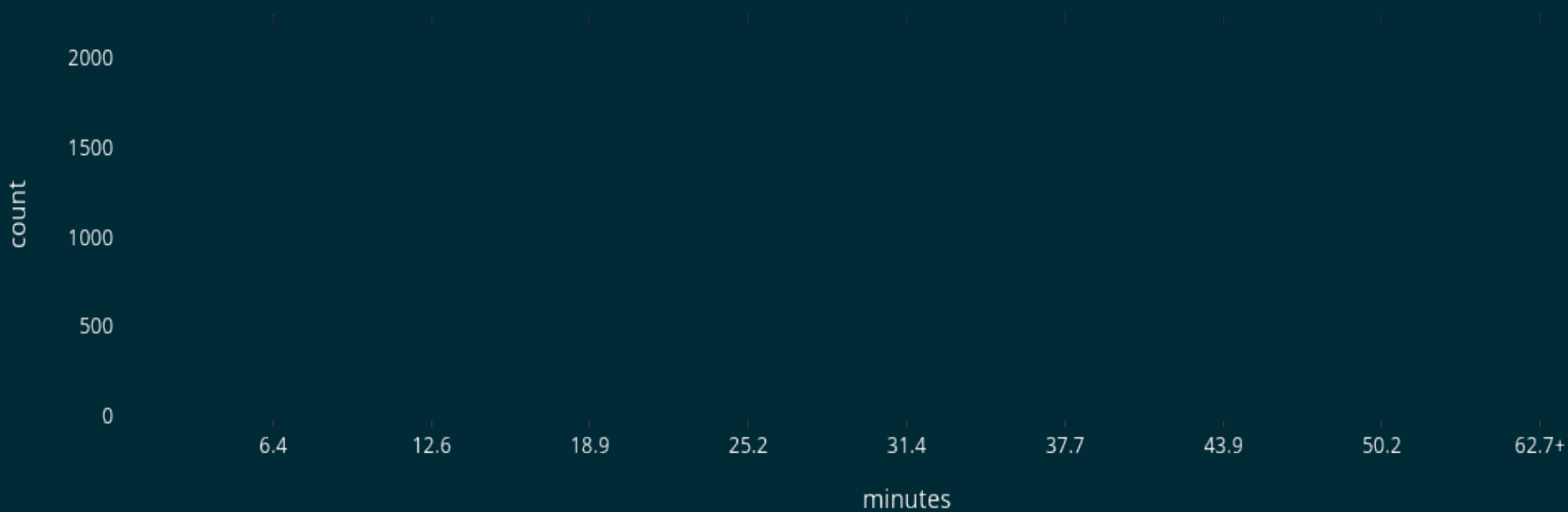
CI Jobs

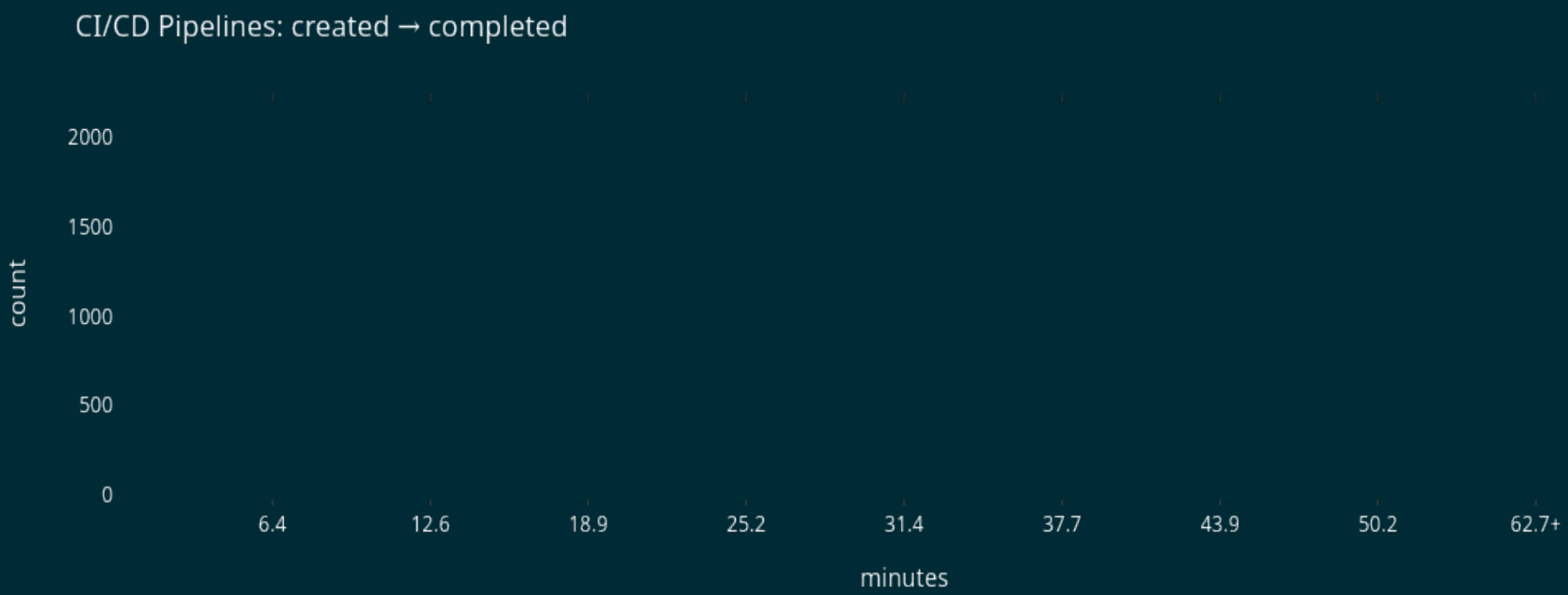
- 58% of MRs: 1+ pipeline
- XX (mean) XX (median) seconds queuing time

CI Jobs

- 58% of MRs: 1+ pipeline
- XX (mean) XX (median) seconds queuing time
- XX (mean) X.X (median) minutes run time (excl. draft MRs)

CI/CD Pipelines: created → completed



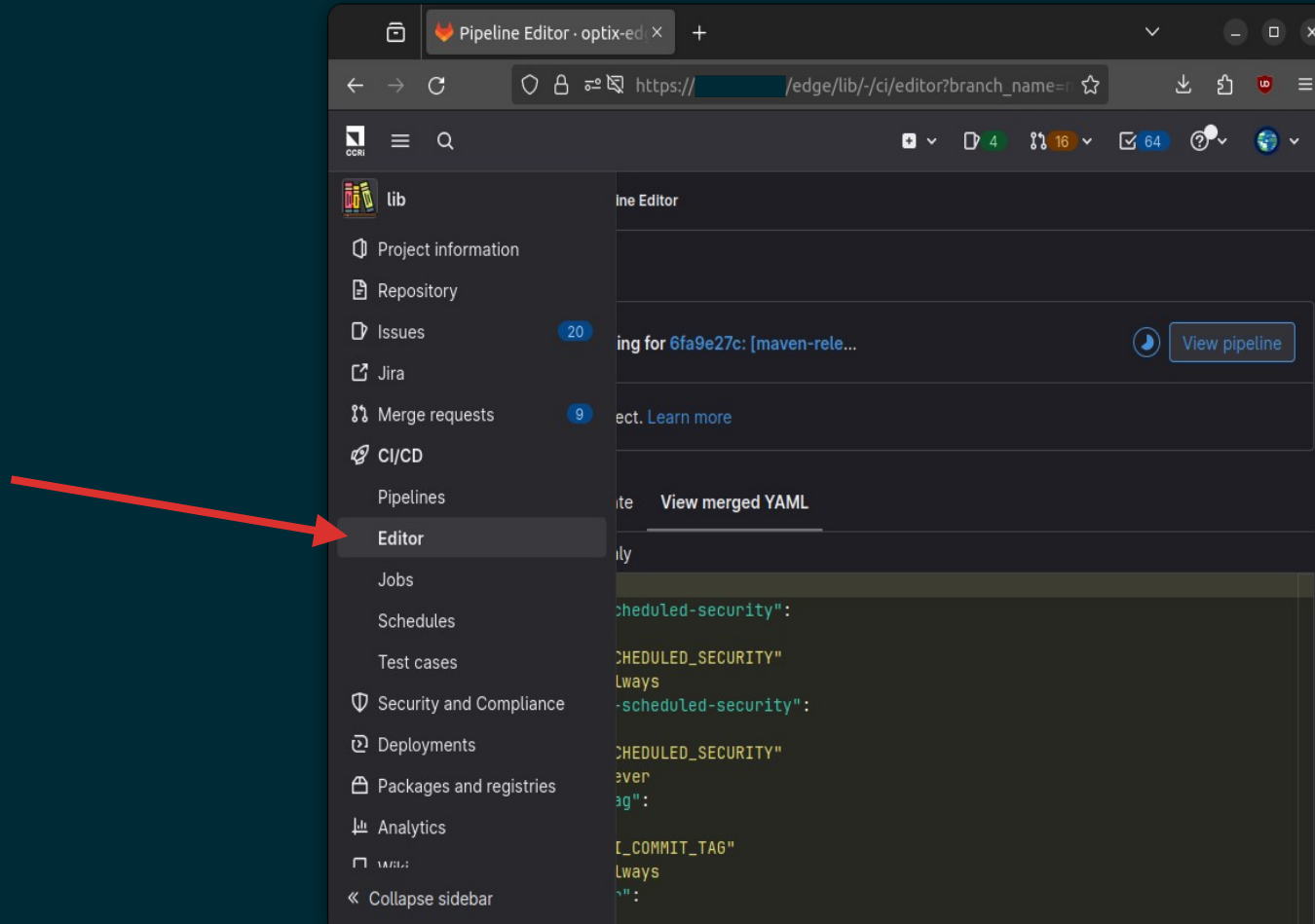


[REDACTED]/hgibson/ci-stats

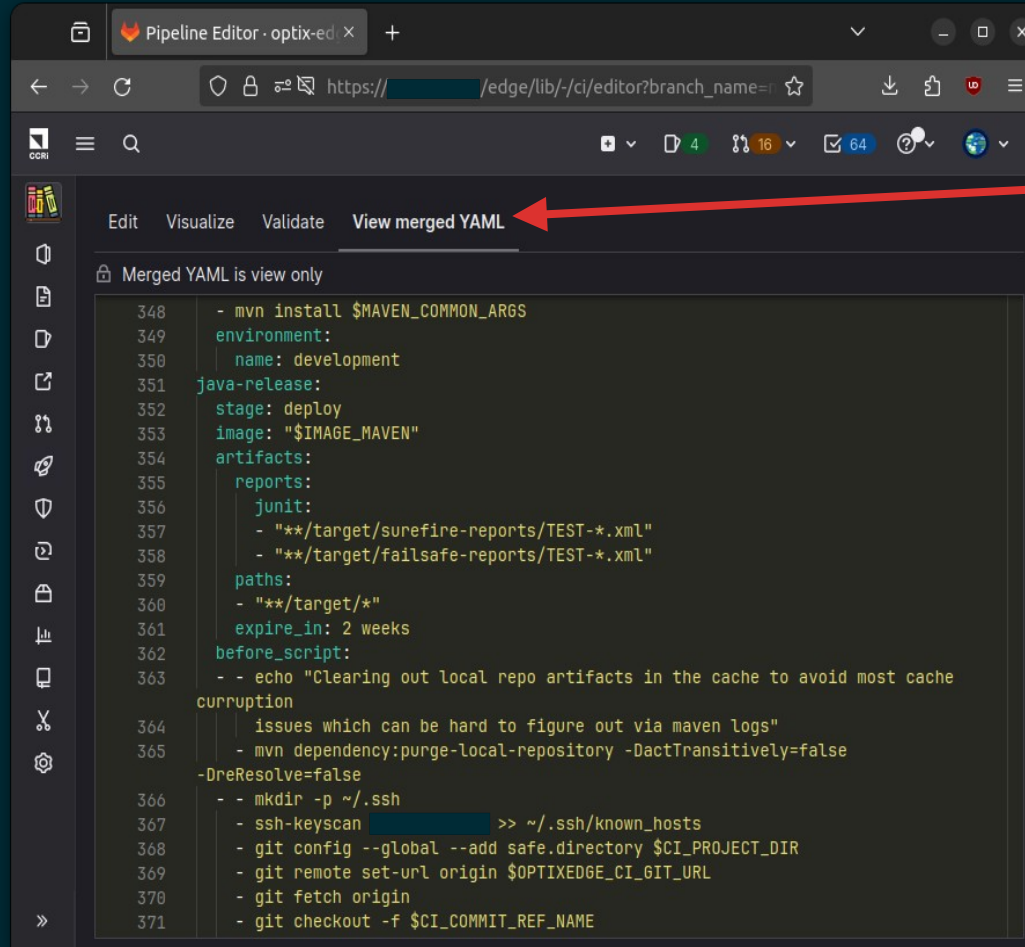
Actionable ideas

The Merged YAML View

The Merged YAML View



The Merged YAML View



Pipeline Editor · optix-ed

https://[redacted]/edge/lib/-/ci/editor?branch_name=r

Edit Visualize Validate **View merged YAML**

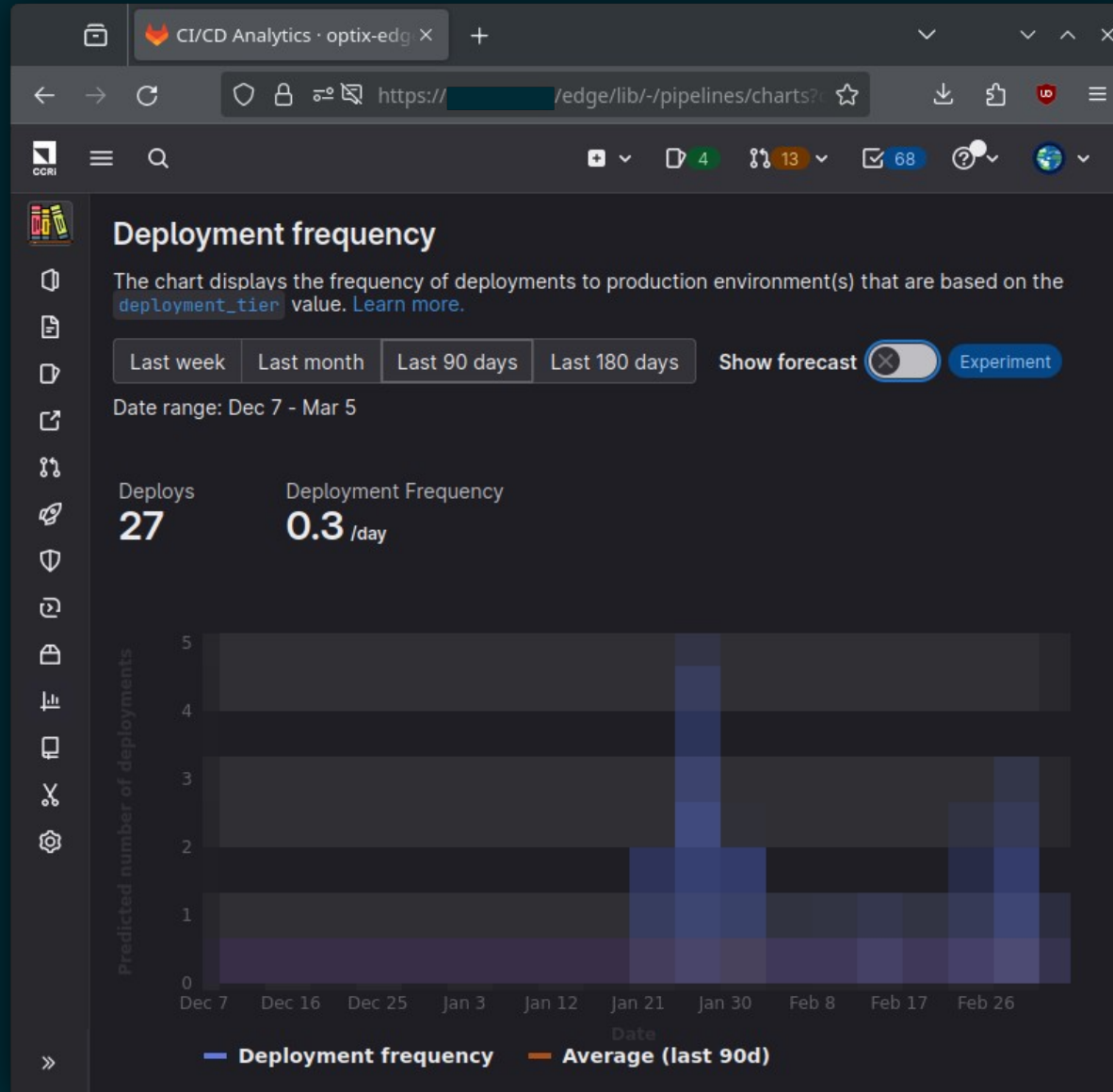
Merged YAML is view only

```
348 - mvn install $MAVEN_COMMON_ARGS
349 environment:
350   name: development
351 java-release:
352   stage: deploy
353   image: "$IMAGE_MAVEN"
354   artifacts:
355     reports:
356       junit:
357         - "**/target/surefire-reports/TEST-*.xml"
358         - "**/target/failsafe-reports/TEST-*.xml"
359     paths:
360       - "**/target/*"
361     expire_in: 2 weeks
362   before_script:
363     - - echo "Clearing out local repo artifacts in the cache to avoid most cache
364       corruption
365       issues which can be hard to figure out via maven logs"
366     - mvn dependency:purge-local-repository -DactTransitively=false
367   -DreResolve=false
368   - mkdir -p ~/.ssh
369   - ssh-keyscan [redacted] >> ~/.ssh/known_hosts
370   - git config --global --add safe.directory $CI_PROJECT_DIR
371   - git remote set-url origin $OPTIXEDGE_CI_GIT_URL
372   - git fetch origin
373   - git checkout -f $CI_COMMIT_REF_NAME
```

Getting changes into production



```
deploy_to_prod:  
  script: ...  
  environment:  
    name: production
```





Provide flexibility
not solutions



```
include:
  - project: "edge/common-cicd"
    ref: "0.13.1"
    file: "gitlab-ci/job-templates/release.yml"
gitlab-release:
  extends: .gitlab_release_job
  stage: deploy
```



```
include:
  - project: 'ccri/ccri-ci-templates'
    ref: '2.4.1'
    file: 'maven/default-maven-build.yml'
variables:
  CI_BUILD: 'false'
  CI_PAGES: 'false'
  CI_GEMNASIUM: 'false'
  CI_CONTAINER_SCAN: 'false'
  CI_LICENSE_SCAN: 'false'
  CI_METRICS: 'false'
  CI_QUALITY: 'false'
```

Adding another job

- > make sure all builds compile

```
include:
- project: "edge/common-cicd"
  ref: "0.13.1"
  file: "gitlab-ci/job-templates/release.yml"
- project: "edge/common-cicd"
  ref: "0.13.1"
  file: "gitlab-ci/job-templates/maven.yml"
- project: "edge/common-cicd"
  ref: "0.13.1"
  file: "gitlab-ci/maven-settings.yml"
maven-build:
  extends: .maven_job
  stage: build
  script:
    - mvn clean compile -DmyParam $MAVEN_COMMON_ARGS
gitlab-release:
  extends: .gitlab_release_job
  stage: deploy
```

```
include:
- project: 'ccri/ccri-ci-templates'
  ref: '2.4.1'
  file: 'maven/default-maven-build.yml'
variables:
  CI_BUILD: 'true'
  CI_PAGES: 'false'
  CI_GEMNASIUM: 'false'
  CI_CONTAINER_SCAN: 'false'
  CI_LICENSE_SCAN: 'false'
  CI_METRICS: 'false'
  CI_QUALITY: 'false'
  SAST_DISABLED: 'true'
  MAVEN_EXTRA_MAIN_ARGS: '-DmyParam'
main_build:
  variables:
    MAVEN_ARGS: 'clean compile'
  before_script:
    # overwrite the before_script
    - 'true'
```

```
include:
- project: "edge/common-cicd"
  ref: "0.13.1"
  file: "gitlab-ci/job-templates/release.yml"
- project: "edge/common-cicd"
  ref: "0.13.1"
  file: "gitlab-ci/job-templates/maven.yml"
- project: "edge/common-cicd"
  ref: "0.13.1"
  file: "gitlab-ci/maven-settings.yml"
maven-build:
  extends: .maven_job
  stage: build
  script:
    - mvn clean compile -DmyParam $MAVEN_COMMON_ARGS
gitlab-release:
  extends: .gitlab_release_job
  stage: deploy
```

```
include:
- project: 'ccri/ccri-ci-templates'
  ref: '2.4.1'
  file: 'maven/default-maven-build.yml'
variables:
  CI_BUILD: 'true'
  CI_PAGES: 'false'
  CI_GEMNASIUM: 'false'
  CI_CONTAINER_SCAN: 'false'
  CI_LICENSE_SCAN: 'false'
  CI_METRICS: 'false'
  CI_QUALITY: 'false'
  SAST_DISABLED: 'true'
  MAVEN_EXTRA_MAIN_ARGS: '-DmyParam'
main_build:
  variables:
    MAVEN_ARGS: 'clean compile'
  before_script:
    # overwrite the before_script
    - 'true'
```

Share templates not jobs

Share templates not jobs

opt-in over opt-out

```
.maven_job: &maven_job
  stage: main
  image: $IMAGE_MAVEN
  artifacts:
    reports:
      junit:
        - "**/target/surefire-reports/TEST-*.xml"
        - "**/target/failsafe-reports/TEST-*.xml"
    paths:
      - "**/target/*"
  expire_in: 2 weeks
  before_script:
    - 'echo "Clearing out local repo artifacts in the cache to avoid most cache corruption issues which can be hard to figure out via maven logs"'
    - 'mvn dependency:purge-local-repository -DactTransitively=false -DreResolve=false'
```

Reliability unlocks performance

Reliability unlocks performance

consistency

Review Speed

Review Speed

- Enforced style

Review Speed

- Enforced style
- Tests = Correctness

Review Speed

- Enforced style
- Tests = Correctness
- Clear review guidelines

Review Speed

- Enforced style
- Tests = Correctness
- Clear review guidelines
- Clear ... everything

Review Speed

- Enforced style
- Tests = Correctness
- Clear review guidelines
- Clear ... everything
 - Who approves?

Review Speed

- Enforced style
- Tests = Correctness
- Clear review guidelines
- Clear ... everything
 - Who approves?
 - What am I allowed to commit?

Review Speed

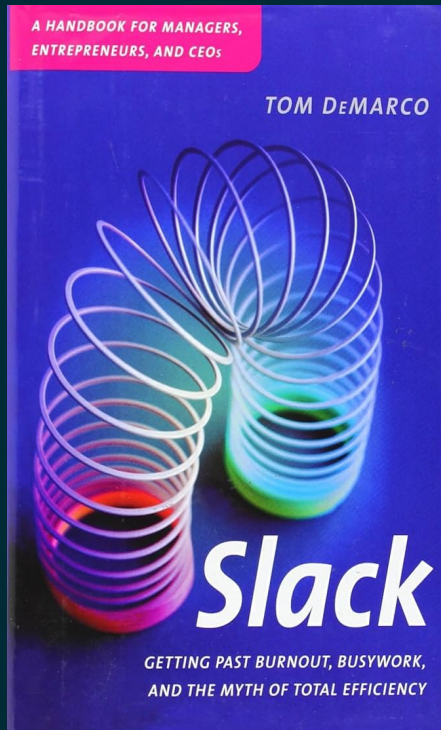
- Enforced style
- Tests = Correctness
- Clear review guidelines
- Clear ... everything
 - Who approves?
 - What am I allowed to commit?
 - What changes do you want?

Review Speed

- Enforced style
- Tests = Correctness
- Clear review guidelines
- Clear ... everything
 - Who approves?
 - What am I allowed to commit?
 - What changes do you want?
 - ...

References

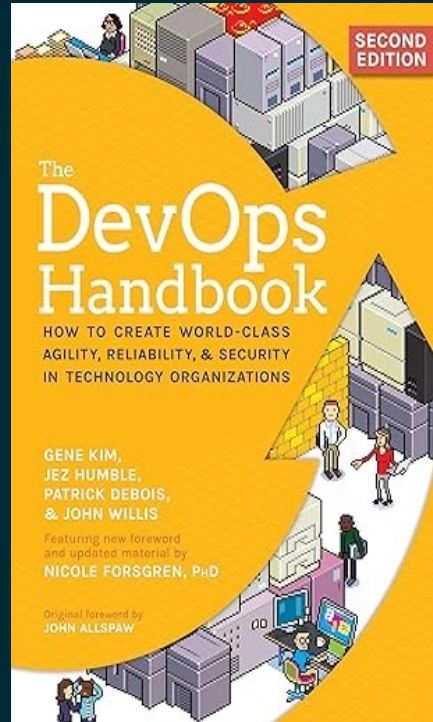
References



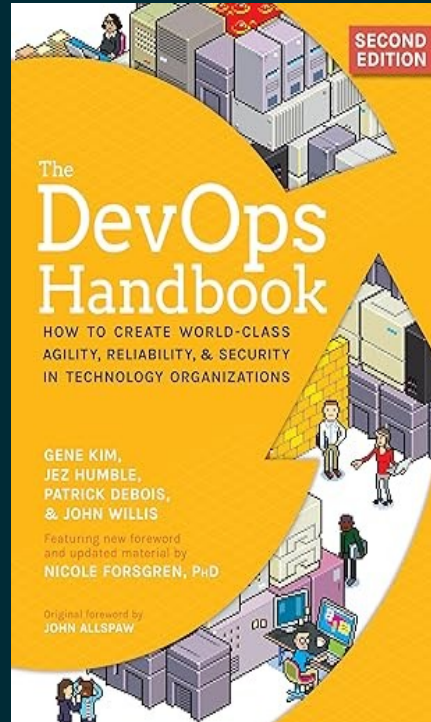
References



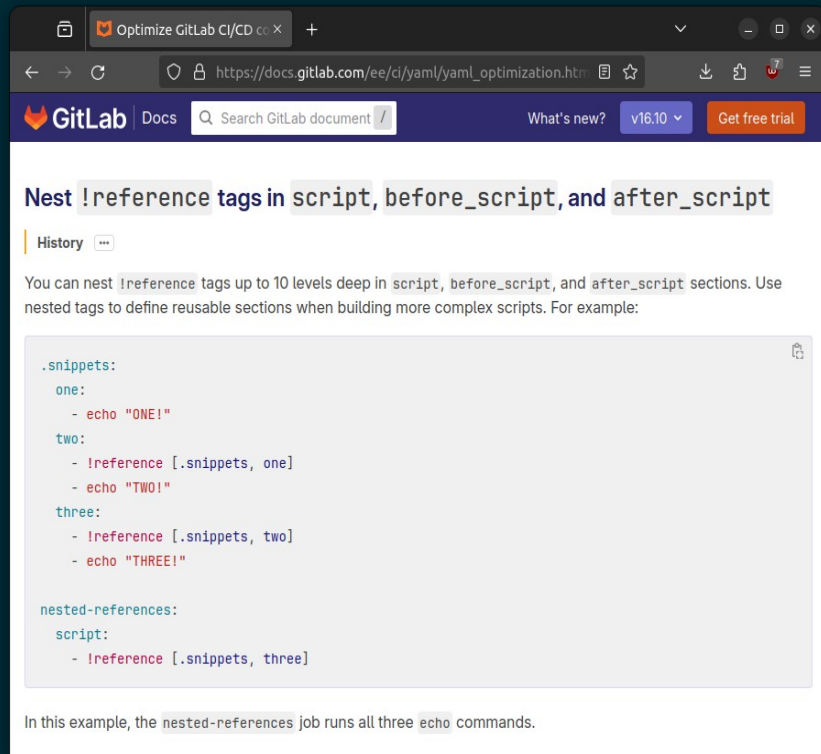
References



References



Cont.



The screenshot shows a web browser window with the GitLab documentation page. The address bar shows the URL `https://docs.gitlab.com/ee/ci/yaml/yaml_optimization.htm`. The page title is "Nest !reference tags in script, before_script, and after_script". Below the title is a "History" section with a dropdown arrow. The main content explains that you can nest `!reference` tags up to 10 levels deep in `script`, `before_script`, and `after_script` sections. An example is provided in a code block showing a `.snippets` section with three items: `one`, `two`, and `three`. `two` references `one`, and `three` references `two`. A `nested-references` section shows a `script` block that references `three`. A note at the bottom states that the `nested-references` job runs all three `echo` commands.

Nest !reference tags in script, before_script, and after_script

History

You can nest `!reference` tags up to 10 levels deep in `script`, `before_script`, and `after_script` sections. Use nested tags to define reusable sections when building more complex scripts. For example:

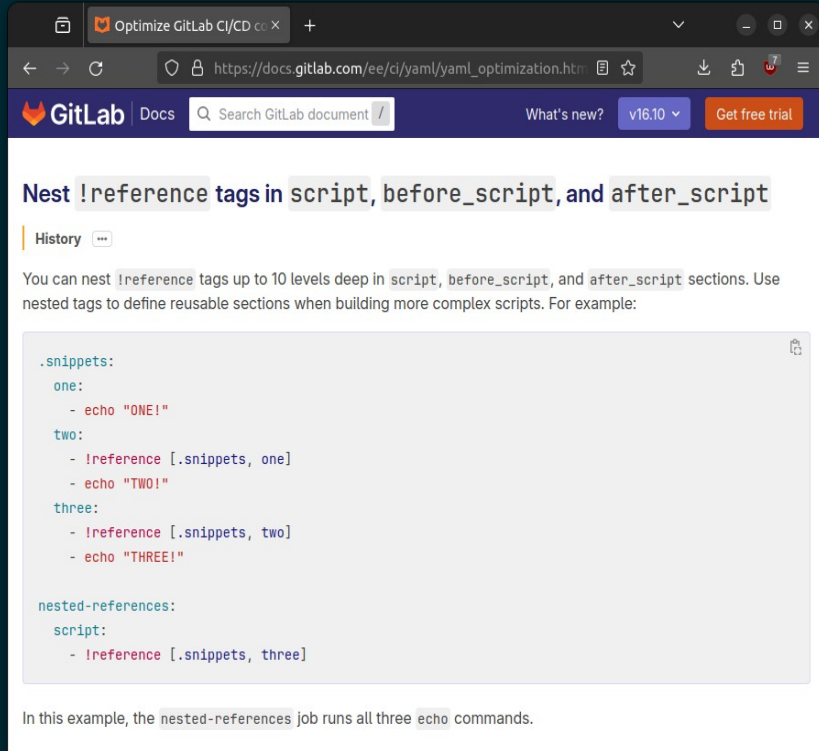
```
.snippets:
  one:
    - echo "ONE!"
  two:
    - !reference [.snippets, one]
    - echo "TWO!"
  three:
    - !reference [.snippets, two]
    - echo "THREE!"

nested-references:
  script:
    - !reference [.snippets, three]
```

In this example, the `nested-references` job runs all three `echo` commands.

docs.gitlab.com

Cont.

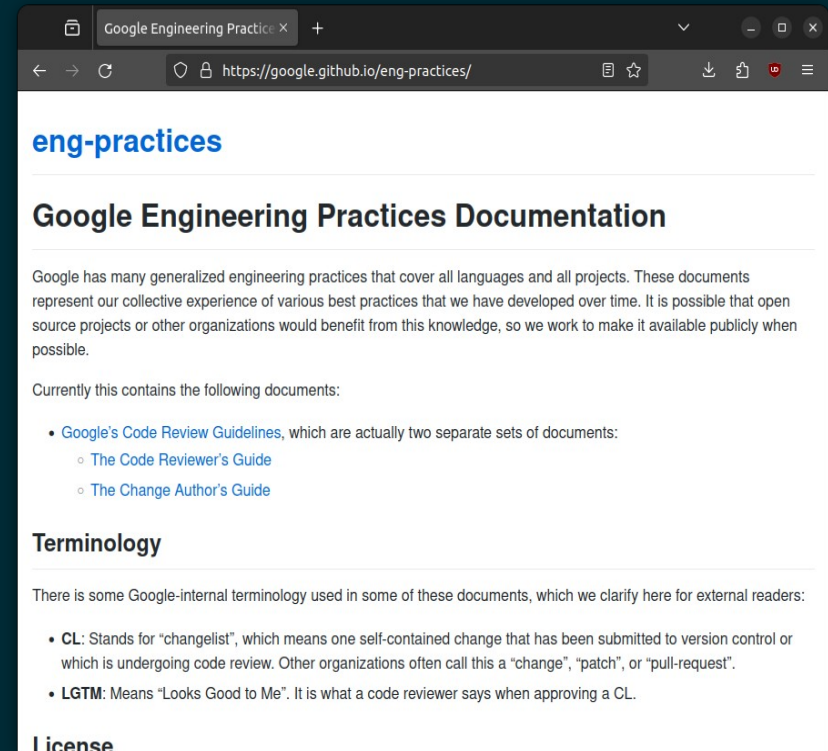


The screenshot shows a web browser window with the URL `https://docs.gitlab.com/ee/ci/yaml/yaml_optimization.html`. The page title is "Nest !reference tags in script, before_script, and after_script". It explains that you can nest !reference tags up to 10 levels deep in script, before_script, and after_script sections. An example is provided showing a .snippets section with three items (one, two, three) and a nested-references section with a script that references the three snippets. A note at the bottom states: "In this example, the nested-references job runs all three echo commands."

```
.snippets:
  one:
    - echo "ONE!"
  two:
    - !reference [.snippets, one]
    - echo "TWO!"
  three:
    - !reference [.snippets, two]
    - echo "THREE!"

nested-references:
  script:
    - !reference [.snippets, three]
```

`docs.gitlab.com/ee/ci/yaml/yaml_optimization.html`



The screenshot shows a web browser window with the URL `https://google.github.io/eng-practices/`. The page title is "eng-practices" and "Google Engineering Practices Documentation". It states that Google has many generalized engineering practices that cover all languages and all projects. These documents represent our collective experience of various best practices that we have developed over time. It is possible that open source projects or other organizations would benefit from this knowledge, so we work to make it available publicly when possible. Currently this contains the following documents:

- [Google's Code Review Guidelines](#), which are actually two separate sets of documents:
 - [The Code Reviewer's Guide](#)
 - [The Change Author's Guide](#)

Terminology

There is some Google-internal terminology used in some of these documents, which we clarify here for external readers:

- **CL:** Stands for "changelist", which means one self-contained change that has been submitted to version control or which is undergoing code review. Other organizations often call this a "change", "patch", or "pull-request".
- **LGTM:** Means "Looks Good to Me". It is what a code reviewer says when approving a CL.

License

Google eng-practices

Introduction | IBM Data Science Best Practices

https://ibm.github.io/data-science-best-practices/

Search IBM Data Science Best Practices

IBM Data Science Best Practices

Introduction

Project Team

Architecture

Source Code

Documentation

Versioning

Data Management

Dependency Management

Configuration Management

Testing

Quality Measurements

Model Training

Distribution

Cloud Deployment

Edge Deployment

Monitoring

Automation

Scaling

Sizing

Security

This site uses [Just the Docs](#), a documentation theme for Jekyll.

IBM Data Science - Best Practices

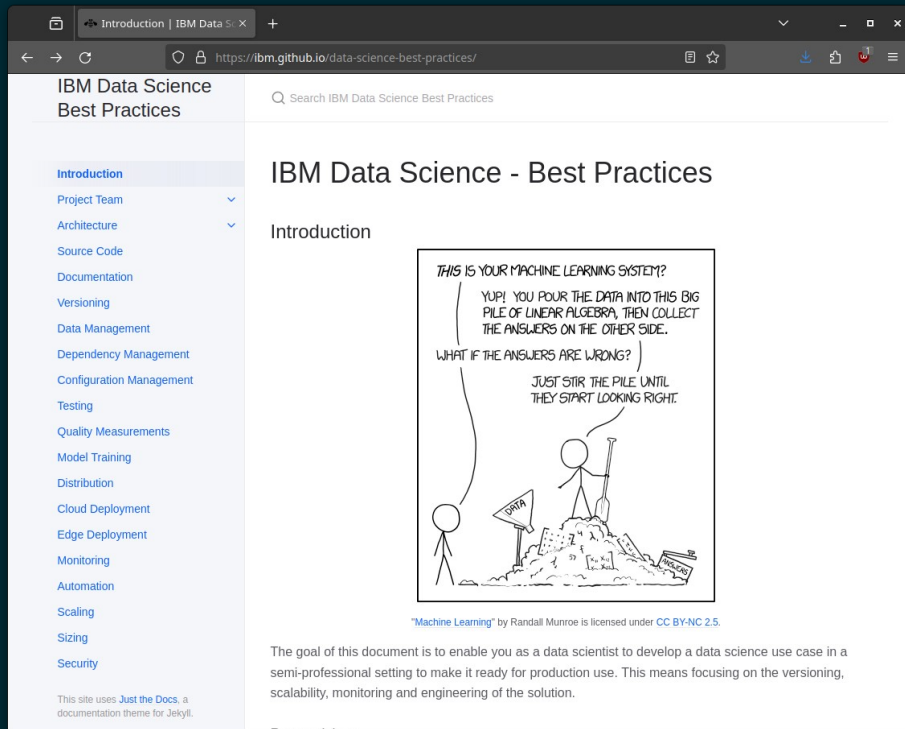
Introduction



"Machine Learning" by Randall Munroe is licensed under [CC BY-NC 2.5](#).

The goal of this document is to enable you as a data scientist to develop a data science use case in a semi-professional setting to make it ready for production use. This means focusing on the versioning, scalability, monitoring and engineering of the solution.

Dependencies



What makes a diff review feel slow?

To answer this question we started by looking at our data. We track a metric that we call "Time In Review," which is a measure of how long a diff is waiting on review across all of its individual review cycles. We only account for the time when the diff is waiting on reviewer action.

The life cycle of a diff

The diagram illustrates the life cycle of a diff as a horizontal timeline. It begins with a blue bar labeled "Diff is published". This is followed by a yellow bar labeled "Author submits a new version". Then comes a blue bar labeled "Reviewer requests changes". This is followed by a yellow bar labeled "Reviewer accepts the diff". The timeline ends with a blue bar labeled "Diff lands into the repo".

Time In Review is calculated as the sum of the time spent in blue sections.

What we discovered surprised us. When we looked at the data in early 2021, our median (P50) hours in review for a diff was only a few hours, which we felt was pretty good. However, looking at P75 (i.e., the slowest 25 percent of reviews) we saw diff review time increase by as much as a day.

Thanks!

Questions?