DEPLOYING WITH CONFIDENCE

WHO AM I

- @gwincr11
- From Wisconsin, California now Pittsburgh.
- Software Engineer at GitHub
- Self Taught Programmer since 14
- Programming professionally 15 years, Ruby, Python, JS, CSS, HTML, Elixir, PHP. Front end and Back.
- Likes Bikes, Soccer, Cooking, programming, Reading when I can
- Parent of 2

TYPES OF DEPLOYS

- Bug fixes
- Incidentals, docs, graphic tweaks
- Feature improvements
- Feature releases
- Data Schema changes
- Roll back deploys

WHAT MAKES CODE RELEASES SCARY?

- Scaling problems?
- Introducing errors?
- Confusing interactions?
- Data changes?
- Missed requirements?
- How multiple releases will effect each other?
- When to merge?

CHALLENGE ACCEPTED

A TOUR OF THE GOOD IDEAS I HAVE EXPERIENCED

- 105 Large news organizations including 16 NBC Universal owned news sites
- A video streaming service and content management system
- A Darpa sigma project entry
- Multi-national bike conglomerate Insera
- A Thoughtbot startup Wootric
- GitHub

GIT-FLOW

https://guides.github.com/introduction/flow/index.html

rails / rails

Used by 1.2m | Watch 2.6k | Unstar 44.3k | Fork 17.9k

Code | Issues 386 | Pull requests 804 | Actions | Security | Insights

WIP: Destroy Associations in Background Job #36912

Edit

Open **gwincr11** wants to merge 16 commits into `rails:master` from `gwincr11:cg-destroy-association-later`

Conversation 25 | Commits 16 | Checks 0 | Files changed 33 | +701 −5

gwincr11 commented on Aug 11 • edited ▾ Contributor

Motivation:

- Sometimes cascading association deletions can cause timeouts due to IO issue. Perhaps a model has associations that are destroyed on deletion which in turn trigger other deletions and this can continue down a complex tree. Along this tree you may also hit other IO

Reviewers

No reviews

Assignees

georgeclaghorn

IT STARTS WITH A PR

GOALS

- Have a consistent way to propose and implement changes.
- Have a clear review cycle.
- Have a long running history of decisions made that is consistent, searchable and audit-able.
- Ensure common rules are enforced with automation.
- Follow Git Flow for all releases.
- Version control for testing different version in different environments.

PR REVIEWS

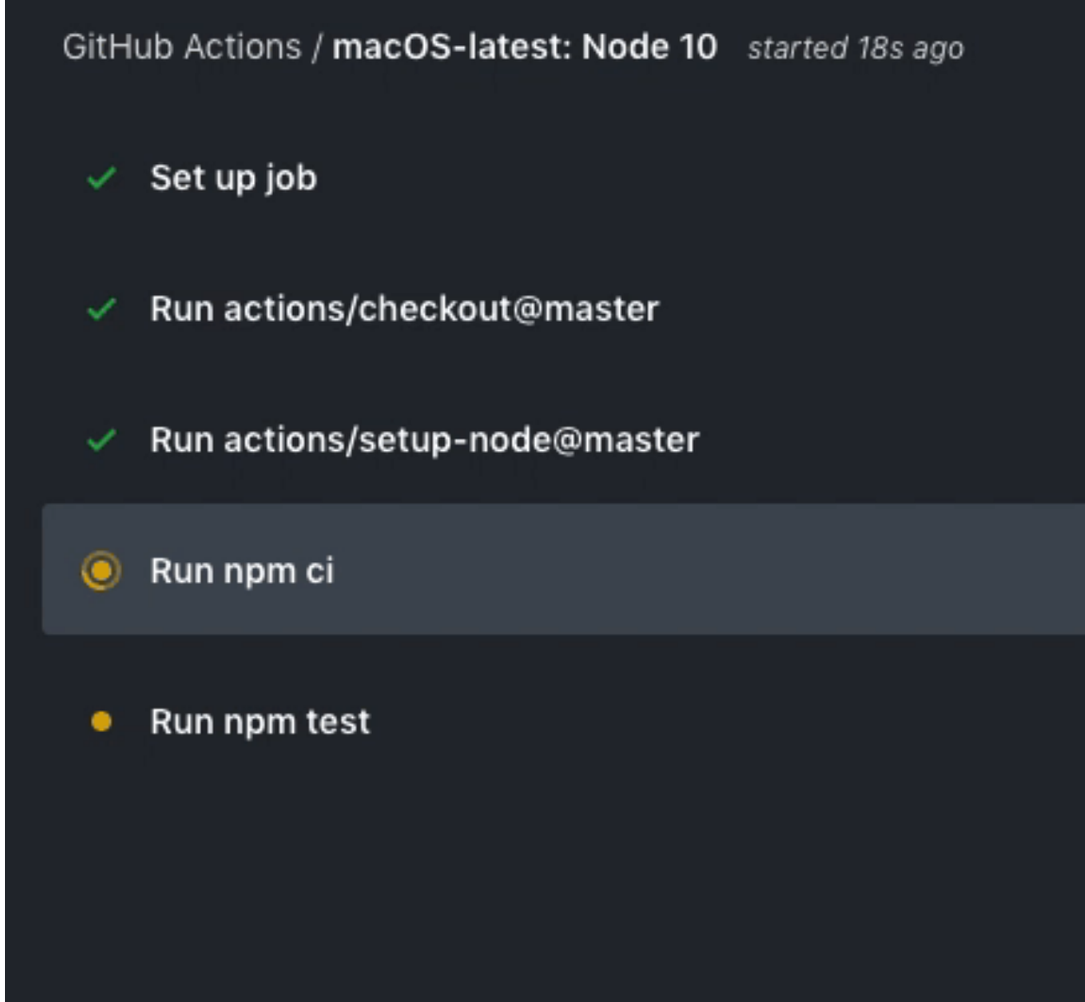
- Branch/fork and then merge. Never commit to master directly.
- Review early and often
- Write draft PR's to signify phase https://github.blog/2019-02-14-introducing-draft-pull-requests/
- Use code owners to ensure proper parties review code changes https://help.github.com/en/articles/about-code-owners
- Lock master until all requirements met.
- Use Bots to check prs.

Reviewers

mtodd

sentinel

iancande


GitHub Actions / macOS-latest: Node 10 started 18s ago
Set up job
Run actions/checkout@master
Run actions/setup-node@master
Run npm ci
Run npm test

TESTING/CI

GOALS

- First line of defense
- Protect against regressions
- Pay it forward
- Guide your implementation
- Think through your implementation before you build it. Tests First.

TESTING / CI

- You cannot CI until you have a good testing culture.
- You can CI without a full test suite.
- Tests should be included with every change.
- All tests should be run in every pr.
- I tend to write integration tests first.
- Check for test coverage.
- There are a ton of opinions here, do what works well for your application.
- Some is better then none.

1 Checks 16 Files changed 5

ils ba9415d

Rails CI / **activesupport**
successful on Aug 8 in 8m 13s

Search

Set up job

```
1 Current runner version: '2.156.4'
2 Prepare workflow directory.
3 Prepare all required actions.
4 action.yml for action: '/home/runner/work/_actions/actions/setup-node/v1/action.yml'.
5 action.yml for action: '/home/runner/work/_actions/actions/checkout/master/action.yml'.
6 Start tracking orphan processes.
```

Initialize containers

Install libraries

Configure databases

Install Node

Install NPM

Checkout

Update Ruby and Dependencies

Install dependencies

Run tests

Stop containers

Complete job

All checks have passed
14 successful checks

group-syncer-fasterer Successful ir

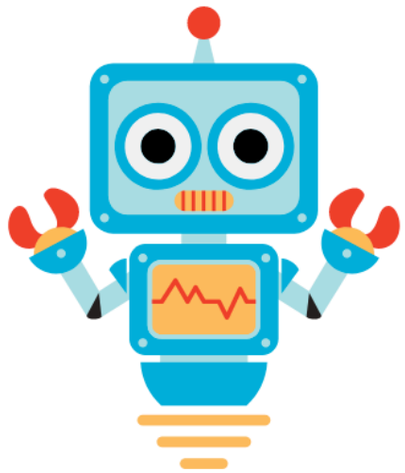
group-syncer-fasterer — Build #1779

group-syncer-lint Successful in 3m

group-syncer-lint — Build #17790716

group-syncer-moda-config-bundle

group-syncer-moda-config-bundle -



Probot

Automate All The Things

AUTOMATE CI/BUILDS CONSISTENCY IN PR'S

- Run test suites on every pr push.
- Run code linters and static analysis on every push.
- Use actions to build/test against different environments.
- Use actions to test deploy step in a sandbox.
- Automate what you can, for security checks, schema alerts and other needs https://probot.github.io/.
- You can use actions to create deployments https://github.com/marketplace/actions/github-deployment.
- https://github.com/gwincr11/example-deploy/actions
- https://github.com/gwincr11/example-deploy/pull/1/files

DEPLOY TO AN AUDIENCE

GOALS

- Limit impact to a group of users you predetermine as safe
- Toggle availability
- Beta test with scale

FEATURE FLAGS

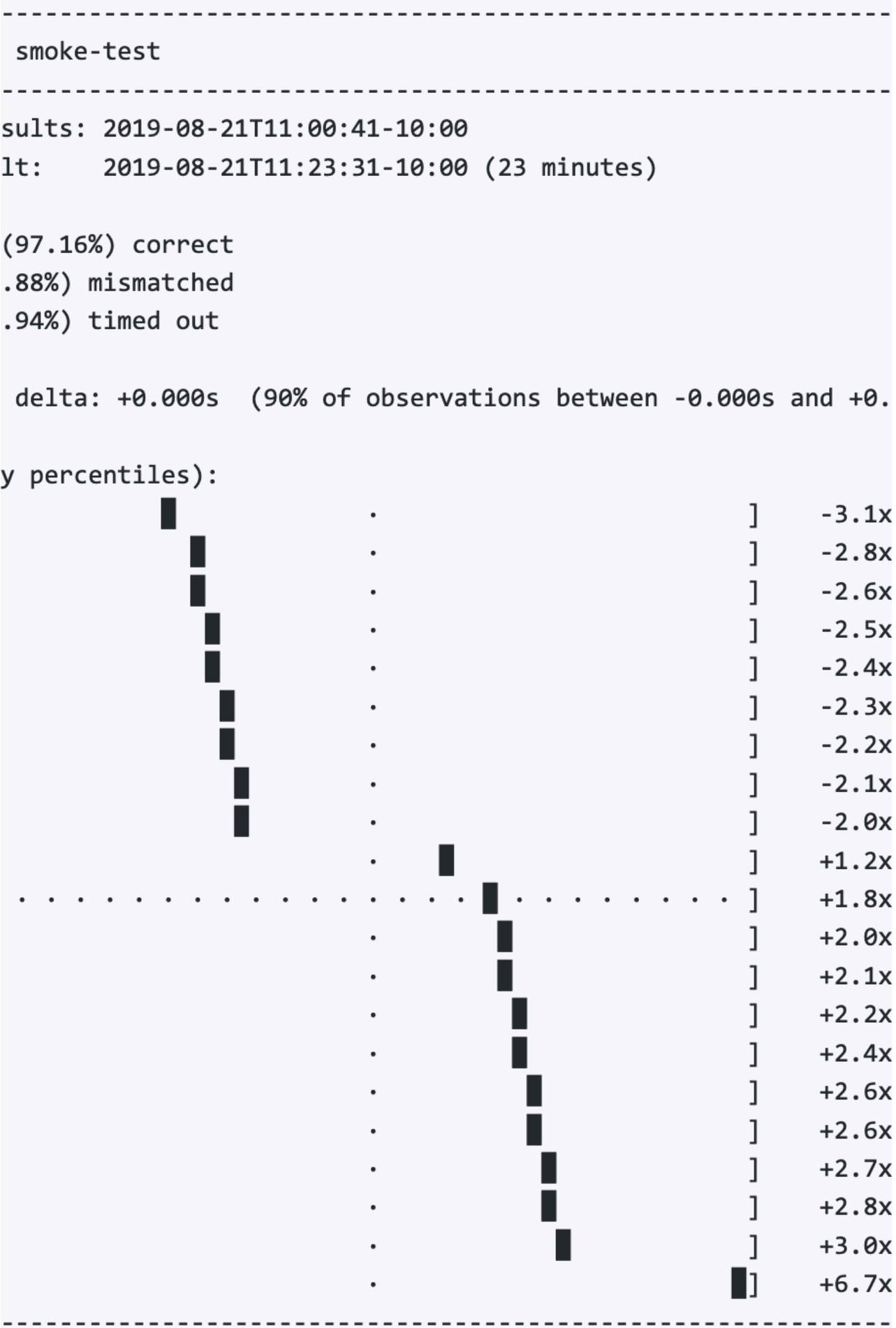
- Limit who has access to a new feature
  - By username
  - By Team
  - By %
  - etc
- We use the flipper gem https://github.com/jnunemaker/flipper
- Make it easy to toggle via ui.

```
------------------------------------------------------------
 smoke-test
------------------------------------------------------------
sults: 2019-08-21T11:00:41-10:00
lt:    2019-08-21T11:23:31-10:00 (23 minutes)

(97.16%) correct
.88%) mismatched
.94%) timed out

 delta: +0.000s  (90% of observations between -0.000s and +0.

y percentiles):
```




```
------------------------------------------------------------
```

EXPERIMENT

- Ensure expected changes or improvements by measuring a subset of requests.
- Try running experiments against friendly group of users.
- We use https://github.com/github/scientist.
- Make it easy to turn off experiments.

HITTING THE BIG TIME

AUTOMATION THE FIRST LINE OF DEFENSE

- Did all the tests pass?
- Did all the linters pass?
- Have the required reviews been completed?
- Is the pr up to date with master? If not update and re-run.

THE QUEUE

- Use tools to automate the queueing of deployments. Create a line of users who want to deploy and enforce the required steps are completed.
  - Order of deployment environments.
  - Time in each.
  - Did exceptions increase?

TEST IN EACH ENVIRONMENT

- Lab deploy, just me.
- Staging deploy, shared env fake data.
- Canary deploy, % of production users.
- Full deploy.
- Merge to master.
- Update all pr’s in the queue.

DEPLOY CONSISTENTLY

GOALS

- Try to keep a running log of every change that has ever happened.
- Have consistent deployment tools.
- All changes need to be reviewed and approved in a consistent manner.
- All changes need to follow appropriate automation pipelines.
- Chatops as logs

CONSISTENCY EVERYWHERE!

- Deploy all changes with PR's as code:
  - DNS
  - Docker changes
  - Security access changes (entitlements)
  - Schema changes, Infrastructure updates/additions

CONSISTENCY OF DEPLOYS

- We use Chatops to deploy everything.
  - Log of all transactions.
  - Consistent deploy syntax
  - All chatops enforce consistent baseline features.
- https://github.blog/2015-06-02-deploying-branches-to-github-com/

WHAT DO DEPLOYS LOOK LIKE

- Hubot (or your slack bot of choice) ->
- GitHub Checks API &&
- GitHub Deployments Api ->
- Heaven (not the open source project but similar) ->
- Capistrano, Heroku, Fabric, Elastic Beanstalk, Custom Shell script ->
- Deployed to env
- http://blog.flowdock.com/2014/11/11/chatops-devops-with-hubot/
- https://developer.github.com/v3/guides/delivering-deployments/
- https://developer.github.com/v3/checks/

GITHUB ACTIONS DEMO

DEPLOYING IS A TEAM EFFORT

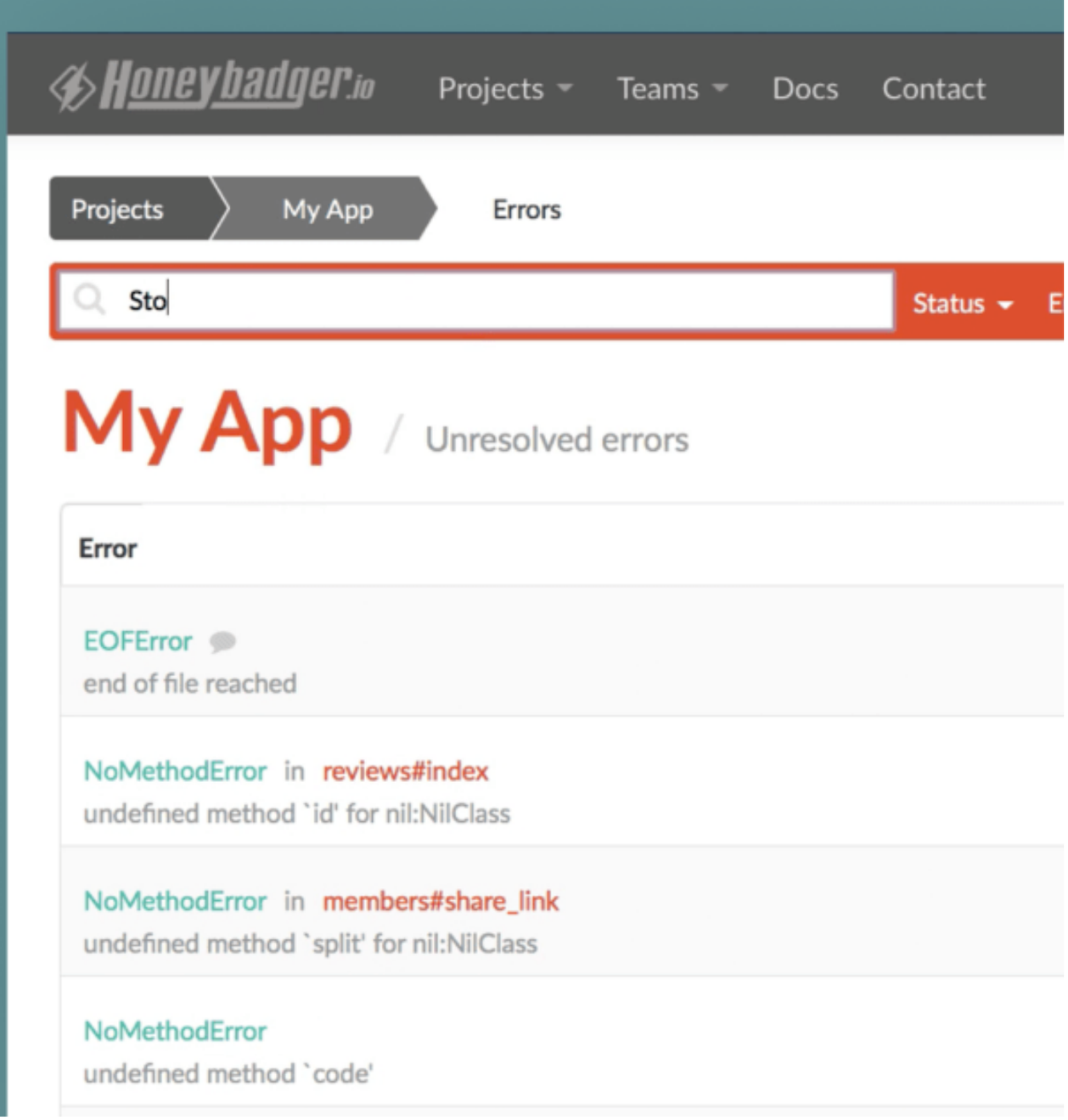
DEPLOYING IS SOCIAL

- Deploy frequently by default, CD. Once it is safe.
- Have a culture of helping and monitoring.
- Uptime is the best feature.
- Have an on call assistant watching deploys.
- Make stats easy to share and access, we have grafana/datadog etc.. chat ops.
- Train deployments.

MONITORING WHILE DEPLOYING

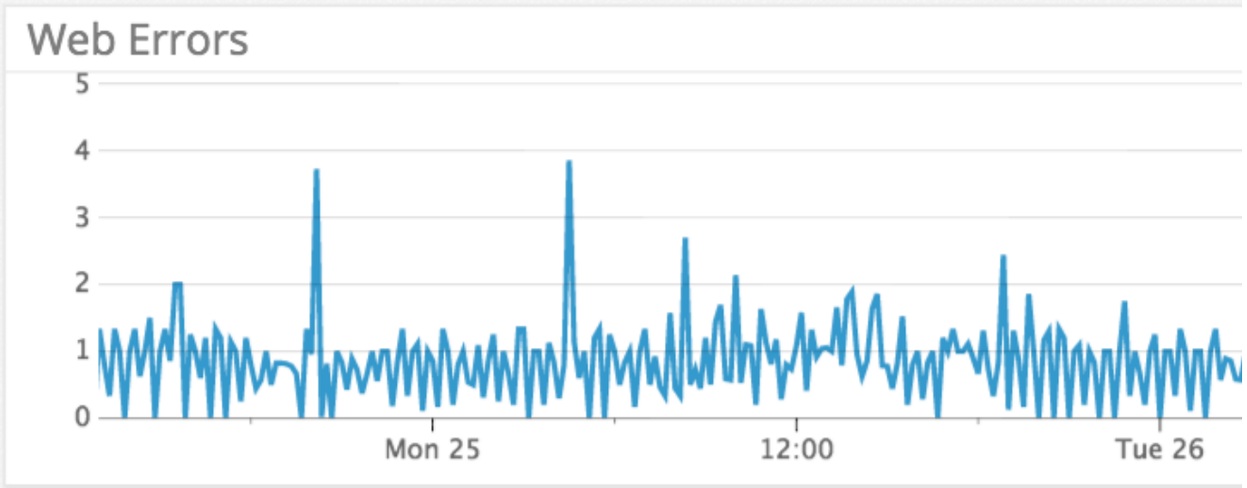
GOALS

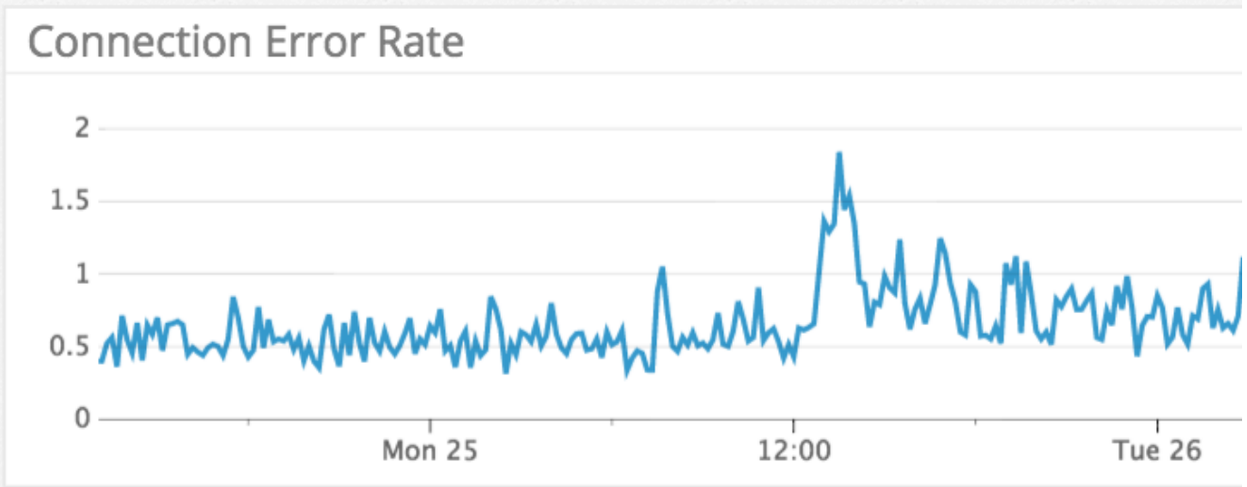
- Provide tools to quickly and easily see common issues.
- Limit the time an issue is in production.
- Limit the users who experience an issue.
- Automate what you can.
- Make it easy to share potential issues with teammates.

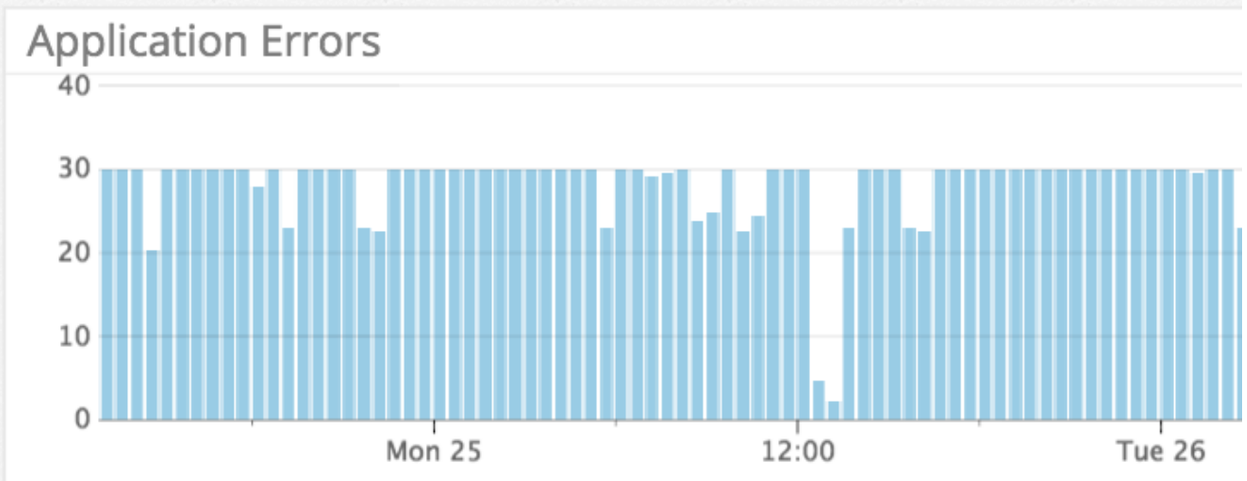


MONITORING EXCEPTIONS

- Make it clear where the error is occurring.
- Where in the app it is occurring, URL, Controller, Endpoint?
- Write custom errors when you can predict things.
- Who is experiencing it?
- Rollup common exceptions.
- Report on most common exceptions.
- Quickly alert the deployer of an increase in exceptions so they don't have to dig.







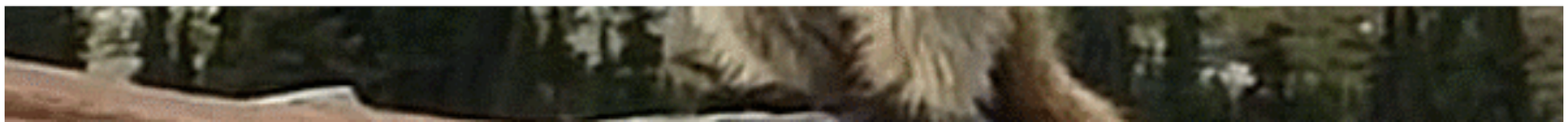
MONITORING PERFORMANCE

- Have a deployment dashboard with common performance indicators for your app
  - Slow queries
  - Memory usage app level, db level and cpu level
  - Job queue
  - Exception counts
  - Response times for all IO
  - Failed responses, 500's, 400's etc…
- Flamegraphs.
- Easily share graphs in chat.

AUTOMATE ALERTING

- Problems are not always caught.
- Automate messaging for common monitor-able issues.
- Broad notification is good as long as it is not noisy.
- Monitoring channels vs. common channels.

SLOWLY INCREASE FOOTPRINTS



GOALS

- Identify issues with the smallest subset of users.
- Limit impact of releases until well tested.
- Validate solutions with only those facing the problem.
- Ensure rollback is easy.
- Encourage risk taking by limiting failure impacts.
- SCIENCE!

DEPLOY TO ENV

- Review labs
- Staging
- Canary
- Production

- Consistent infrastructure across all.

GOOD CANARY RULES

- Ensure the thing being deployed is tested with significant traffic.
- For critical service deploy more canaries for longer
- Have a separate health dashboard for monitoring canaries
- Cover 5-10% of the workload
- It should be safe for canaries to fail.
- Include more than one instance when possible.
- https://martinfowler.com/bliki/CanaryRelease.html

CHANGING DATA

GOALS

- Non-Destructive.
- Rolling changes.
- Deploy separately from code.
- Allow for long running data changes.
- Easy to rollback.

SCHEMA CHANGES

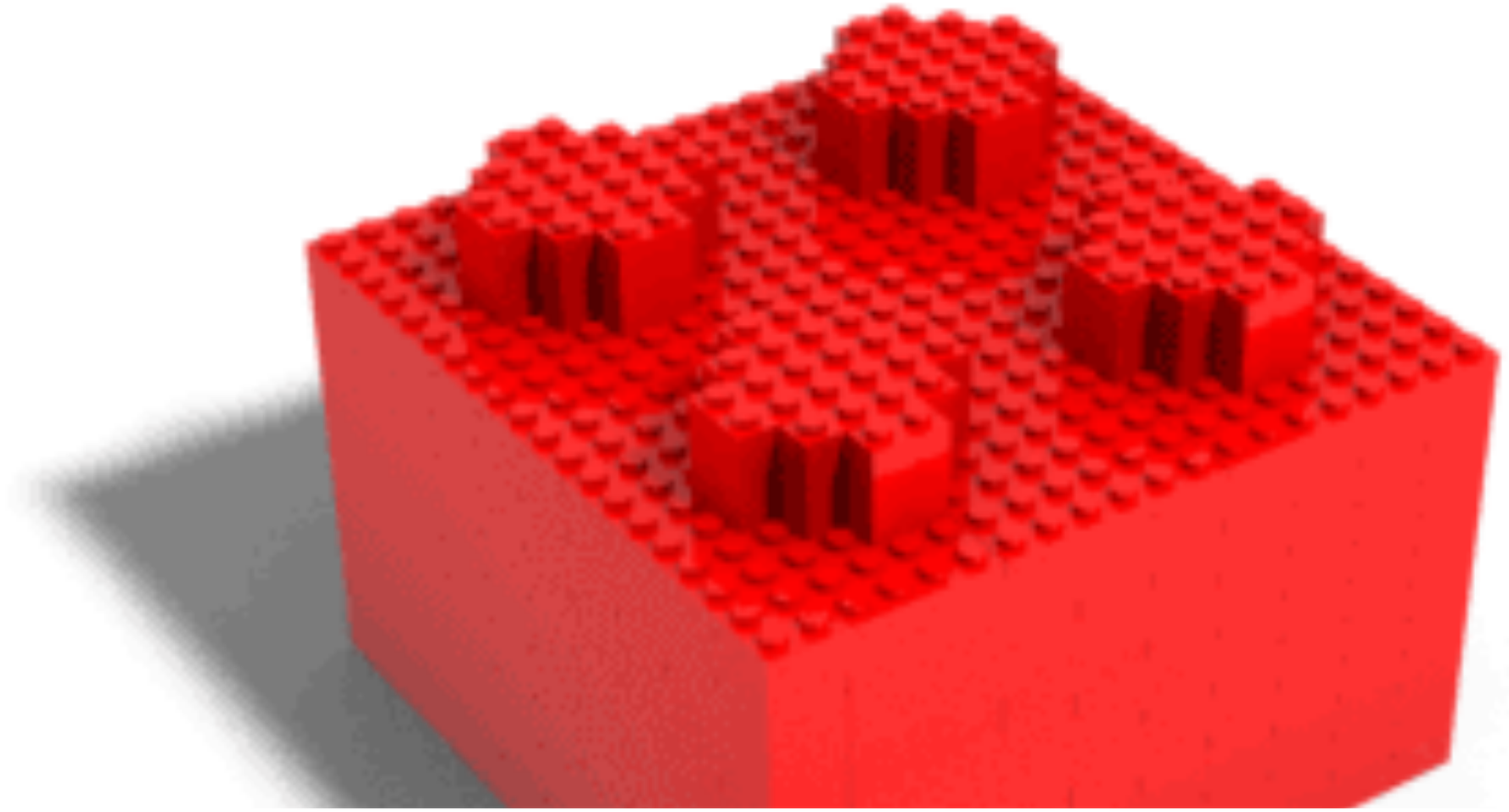
- Separate migration pr from code changes.
- Backwards and forwards compatible, in case of rollbacks, with code and schema changes.
- One pr for additions, separate pr for removals after it is safe to remove old data.
- Require separate reviews.
- Run in test environment first.
- If dropping columns uses tools to ensure nothing is utilizing the column.
- Ensure all the indexes you need are present.

RENAMING A COLUMN

- Add a new column with the new name.
- Double write to the old and new column to capture new writes.
- Write a transition to copy the old column to the new column.
- Once everything is staying in expected state drop the old column.

DATA TRANSITIONS

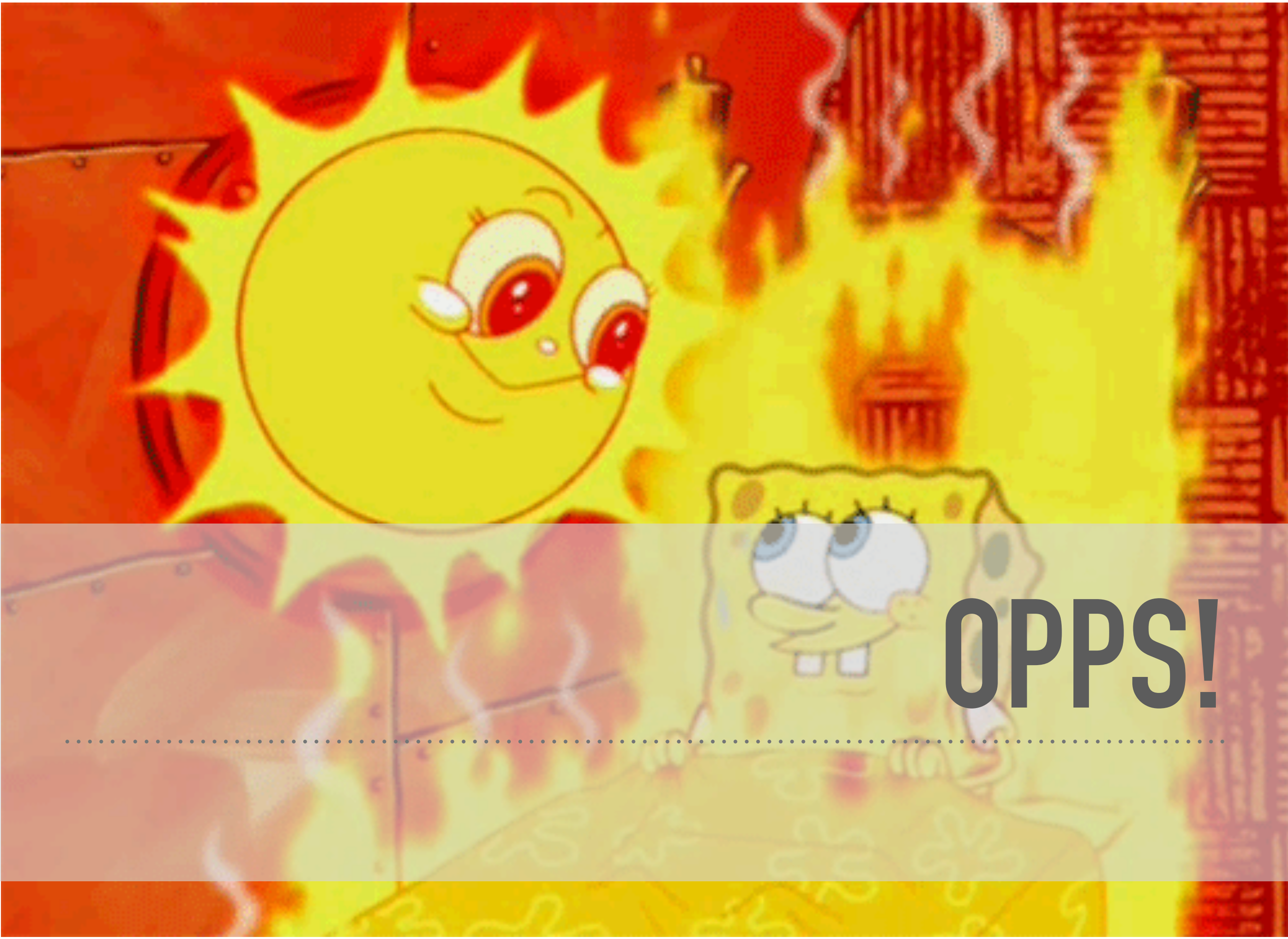
- Separate pr's from data changes or code changes.
- Run in test environment first.
- Beware of altering moving data.
- Beware of changing indexes and slow query implications.
- Add indexes to make transitions fast.
- Use MySQL batches instead of trying to update everything at once.
- Use Read Only Replicas when possible for lookups to reduce load on write databases.
- Create a dry run transition to check state before running.
- Add logging.
- Write a test for your transition.



IT'S A FEATURE

DEPLOYMENT IS A FEATURE

- Who owns this process?
- What are your best practices?
- How do you enforce consistency?
- How do you enable quick analysis and action in case of a bad deploy?
- Is your process blameless?
- What repetitive tasks can you automate?
- What 1 thing can you do to feel more confident?



OPPS!

WHEN IT GOES WRONG

- Availability is the best ability.
- Re-Deploy master.
- Revert PR's.
- Non-Destructive data changes.
- Backups.
- Blameless retros
- What was learned and what can be better?
- You are responsible for the mental health of your team, be kind.

THANK YOU!