AWS re:Invent

DOP205-S

Amplifying CI/CD helped Wag! reduce release process by 600%

Brandon Jung
VP Alliances
GitLab



Dave Bullock
VP of Technology,
Engineering & Operations
Wag Labs, Inc.

AWS re:Invent

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Business survival depends on a radically faster DevOps lifecycle



"Cycle time compression may be the most underestimated force in determining winners & losers in tech."

—Marc Andreessen

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

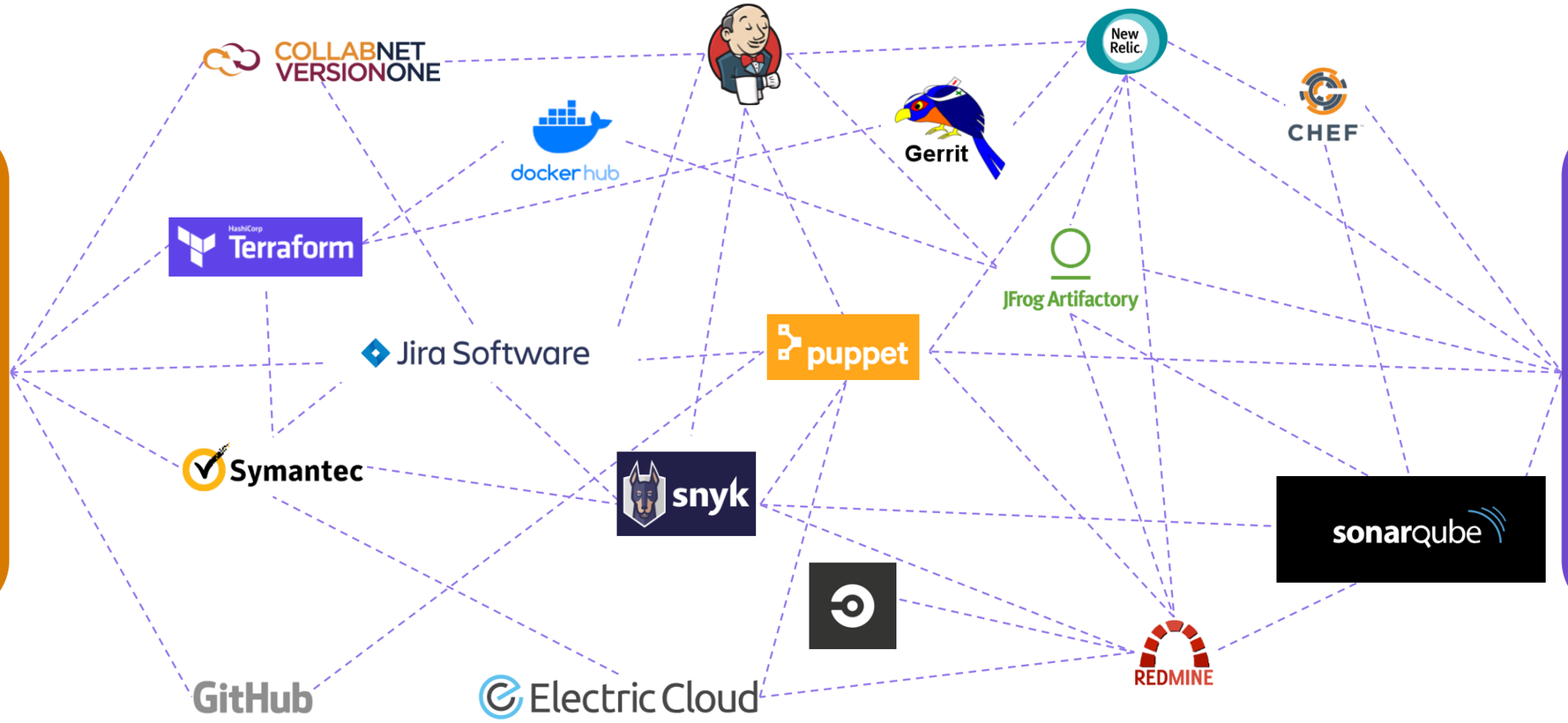
Today's toolchain **limits** a faster DevOps lifecycle



Complexity slows down DevOps initiatives

Problems

- Keep up with the competition
- Understand performance and business drivers
- Improve efficiency
- Protect business from security threats



Solutions

- Launch new features, expand product offerings, and roll out upgrades
- Use data insights to benchmark performance and assess peers
- Automate processes
- Monitor, discover, and address security breaches

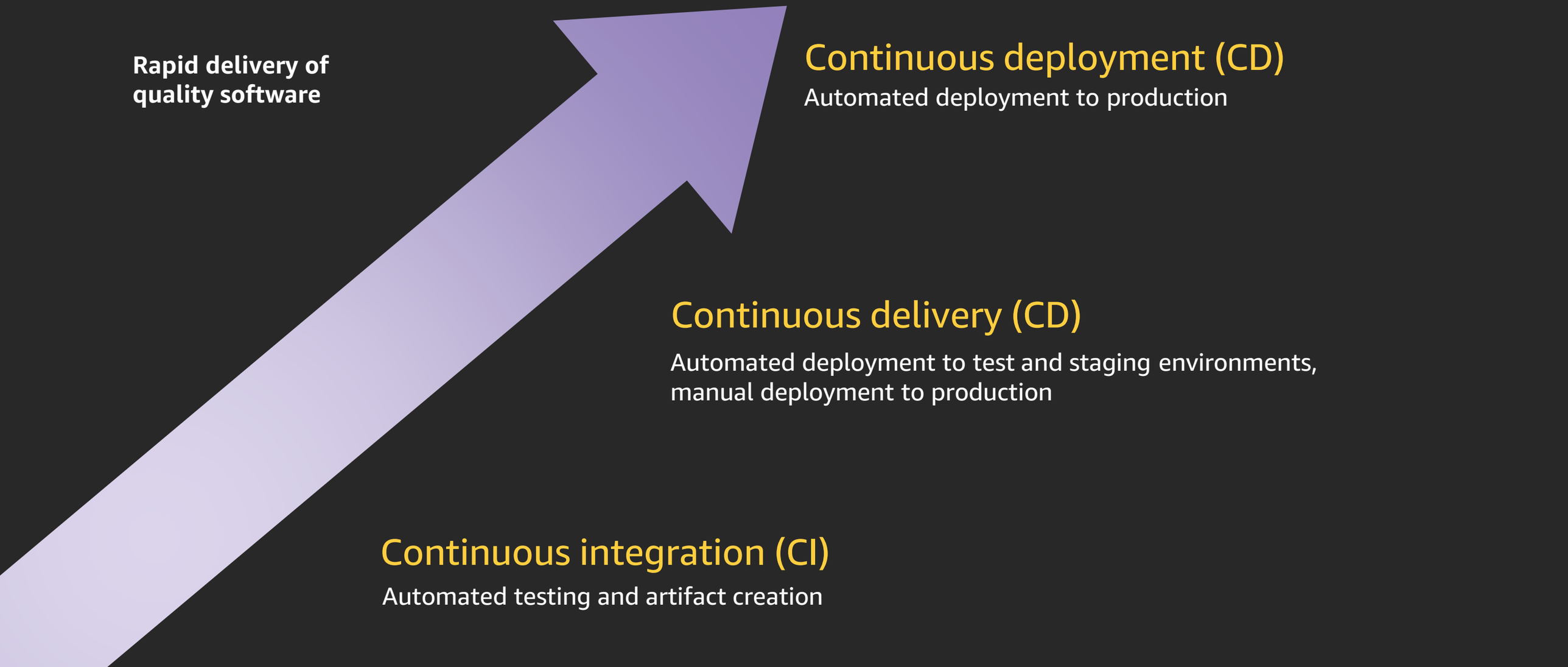
X 100s of tools

X Multiple data models

X Complexity & risk

X Lack of transparency

Complexity slows down DevOps initiatives



So many options… where should you begin?

Continuous integration



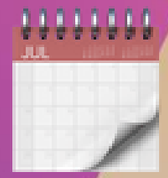
Continuous delivery/deployment

Wag! intro

- On-demand dog walking, sitting, boarding, and daycare platform
- In 43 states and over 110 cities
- Millions of walks per year
- We create joy for dogs and those who love them
- We like dog puns
- Slackmoji-heavy environment

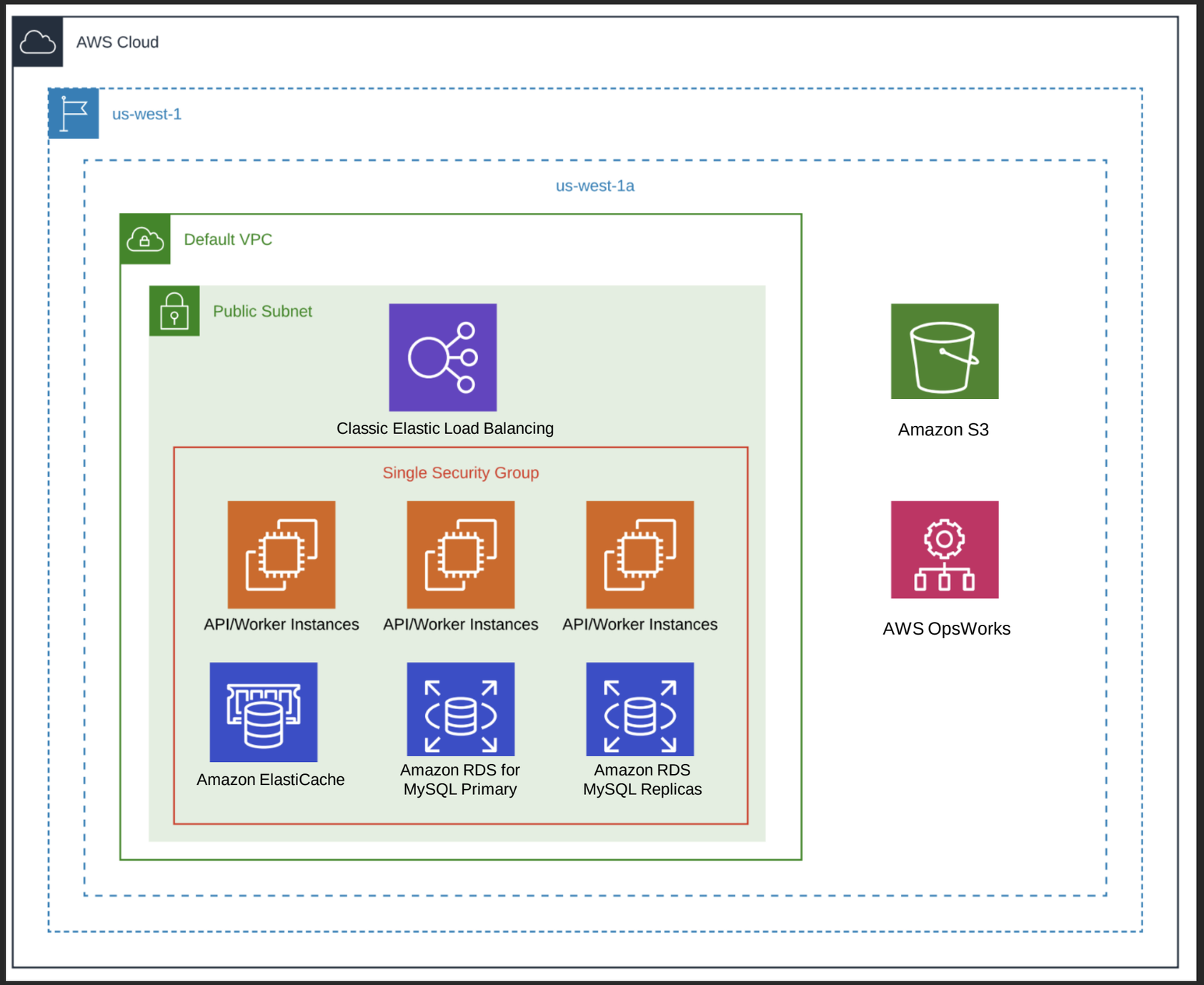


1.5 years ago



© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What things looked like before



The problem (a)

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The problem (a): Application pipeline

- Crusty, brittle, complicated, and dangerous deployments for our service
- Slow recovery path for rollbacks
- Divergent environments – development vs. stage vs. production
- Limited, fixed-sized pipeline for CI with Travis CI
  - Bottleneck half the day
  - Wasting money the other half
- Single point of contact/failure for releases across the entire organization

The plan: Phase 1a

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The plan: Phase 1a

PAW-tomate all the things!

- Containerize the application
- Automatically scaling Amazon Elastic Container Service (Amazon ECS) cluster
- Blue/green + canary deployments
- Democratize deployments



Containerize the application

- Build out pipeline in GitLab with Ansible and Packer
  - Creates base Docker and AMIs nightly
  - Latest versions of the base OS/packages/etc.
  - Trigger build anytime to push security patches
  - Pushes base image into Amazon Elastic Container Registry (Amazon ECR)
- Build out stage in GitLab that pulls down base image from Amazon ECR
- Copy branch/tagged release into Docker image, push back to Amazon ECR and tag
- No secrets or config in the image
  - Same image used for automated tests, QA, staging, and production

Amazon ECS cluster

- Deploy application into an Amazon ECS cluster
- Pull in secrets and config using SSM and Chamber
- CPU-based scaling
- Saved money and increased scalability

Blue/green + canary deployments

- Spin up a new Amazon ECS cluster before deployment
- Pull in tagged release image from Amazon ECR
- Scale out cluster to match live cluster
- Send 1% of traffic to new cluster
- Monitor logs/errors/load/etc.
- Cut over to new cluster
- Monitor logs/errors/load/etc.
- Scale down or roll back
- Implemented with Amazon Route 53 + layer of HAProxy

Democratize deployments

- Empower every team to deploy its own code
  - Simple web interface
    - Database migrations
    - Canary
    - Cutover
    - Rollback
- Foster end-to-end ownership of the complete system
- Increase release velocity
- Reduce risk

The problem (b)

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The problem (b): Infrastructure

- **IaCAtAWSC** (infrastructure as clicking around the AWS Management Console)
- No automatic scaling
- Flat, default VPC, default subnets, no network ACLs
- No staging environment for testing changes
- One security group to rule them all



The plan: Phase 1b

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The plan: Phase 1b

Rearchitect infrastructure

- Define as-code
- Automatic scaling, HA Multi-AZ architecture
- Tiered VPC subnets
- Minimally permissive network ACLs/SGs
- Enable application deployments via GitLab

Define as-code

Rearchitect entire infrastructure

- Code, not clicks
- Terraform all the things
- Built a big monorepo for modules/environments (only 1 dev)*
- Amazon S3 for state and Amazon DynamoDB for locking (one per environment)*
- Nothing hard-coded (/shrug mostly)

*ended up having to refactor these!

Automatic scaling HA: Multi-AZ

- Everything was built to be *n*-AZ
- Sandbox/testing Single-AZ
- Production 2 + Availability Zone
- No single points of failure
- Automatic scaling with variable instance sizes
- Sandbox and stage with same architecture as production, but at a smaller scale :money_with_wings:

Tiered VPC

- Subnets for every tier
  - APIs, workers, etc.
  - Database
  - Cache
  - Application Load Balancer (ALB)/NAT gateway/etc. (only public subnets)
- Lock down network ACLs, minimally permissive between tiers
- Security groups to match, add stateful rules
- Turn on flow logs

Enable deployments

- Build out Terraform to perform aforementioned blue/green + canary deployments
- Use GitLab runners with AWS Security Token Service (AWS STS) role assumption to do cross-account deployments
- No need to log in to systems, move code, run commands, etc.
- All handled via Terraform

A few months later



© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

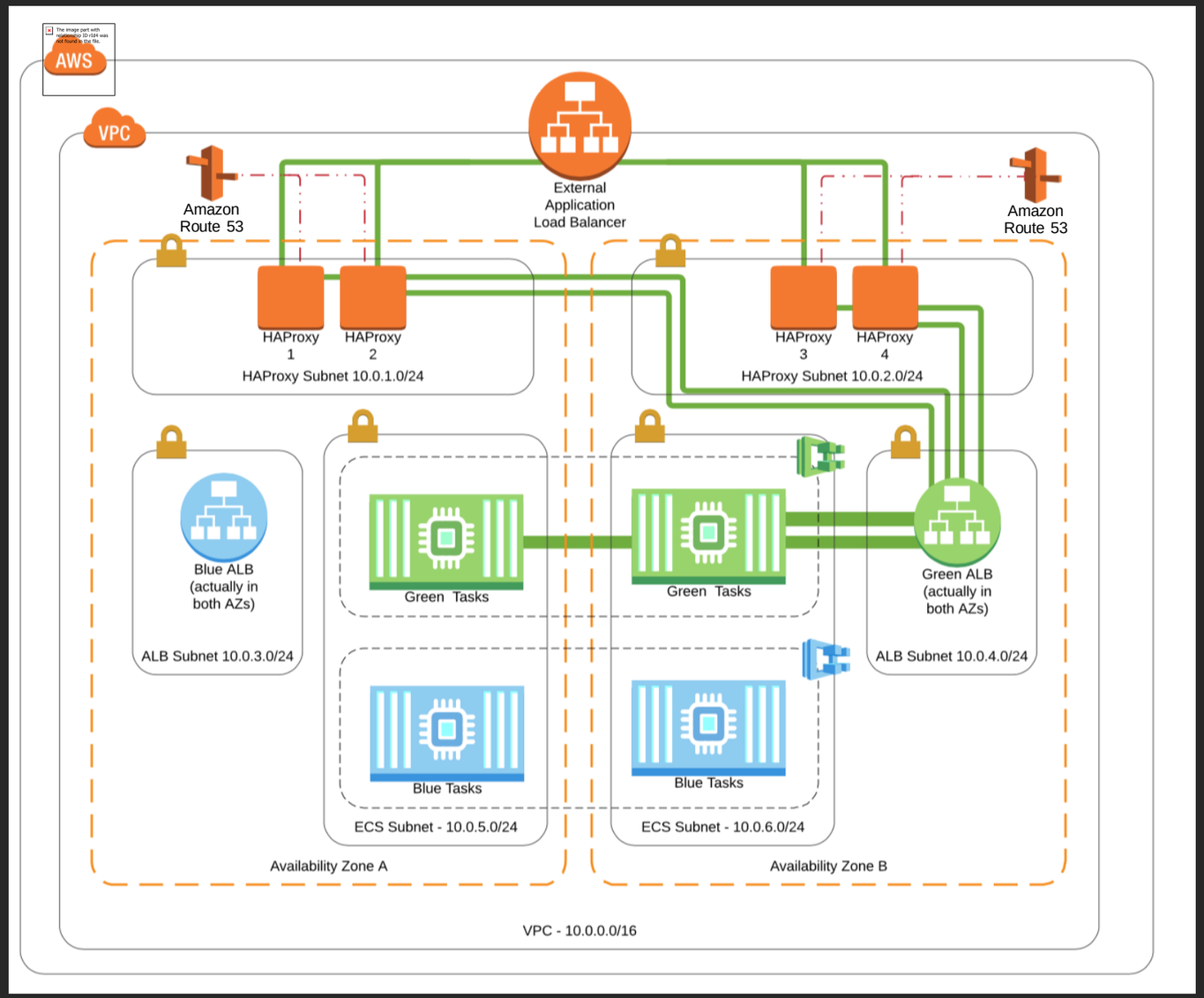
Success*

- Piece by piece, we moved our entire platform to our new environment
- We empowered the teams to do their own deployments
- Systems were scalable, locked down, etc.
- Passed AWS Well-Architected review
- Saved money
- Increased release velocity
- Reduced deployment risk

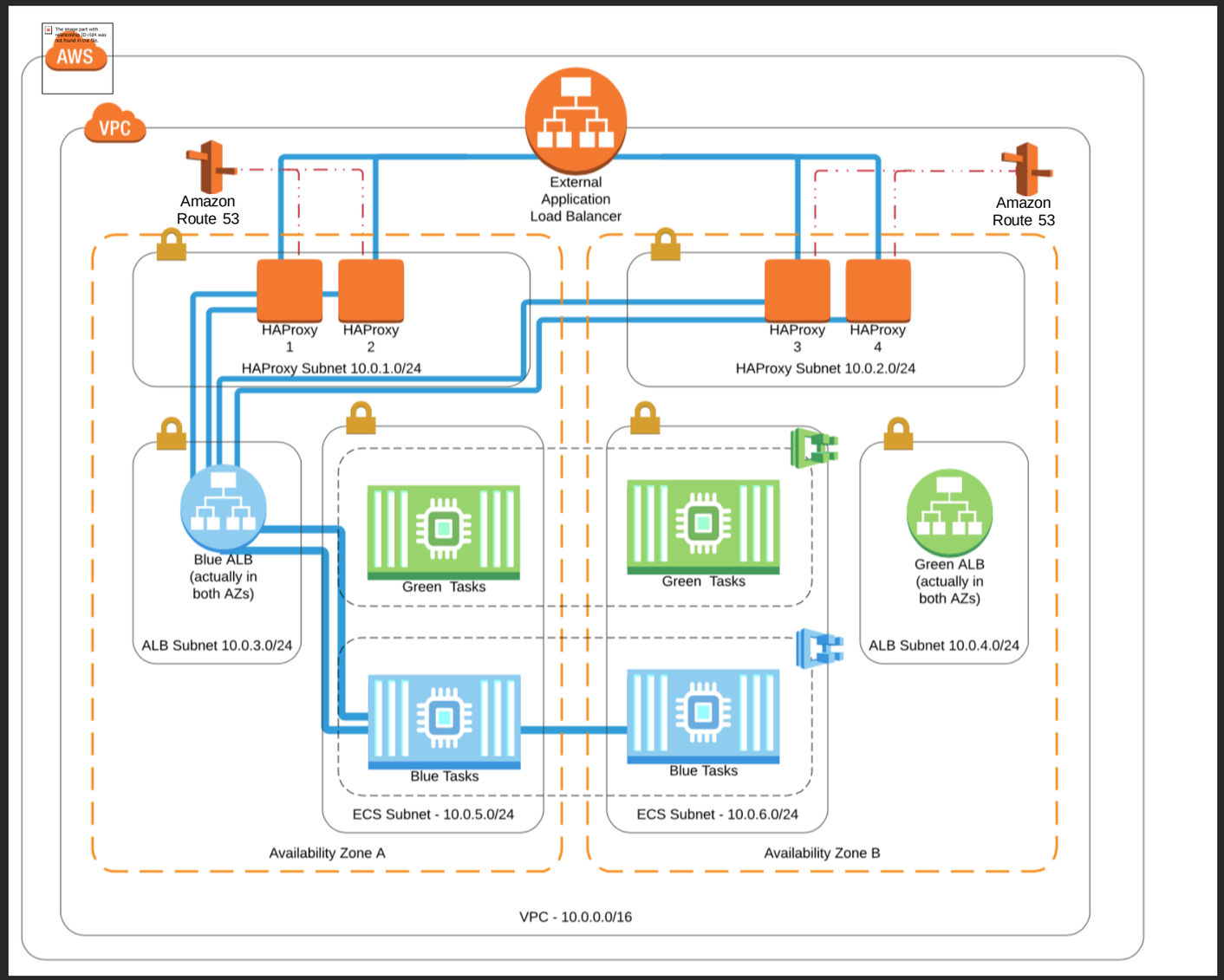
*Not everything was sunshine and rainbows



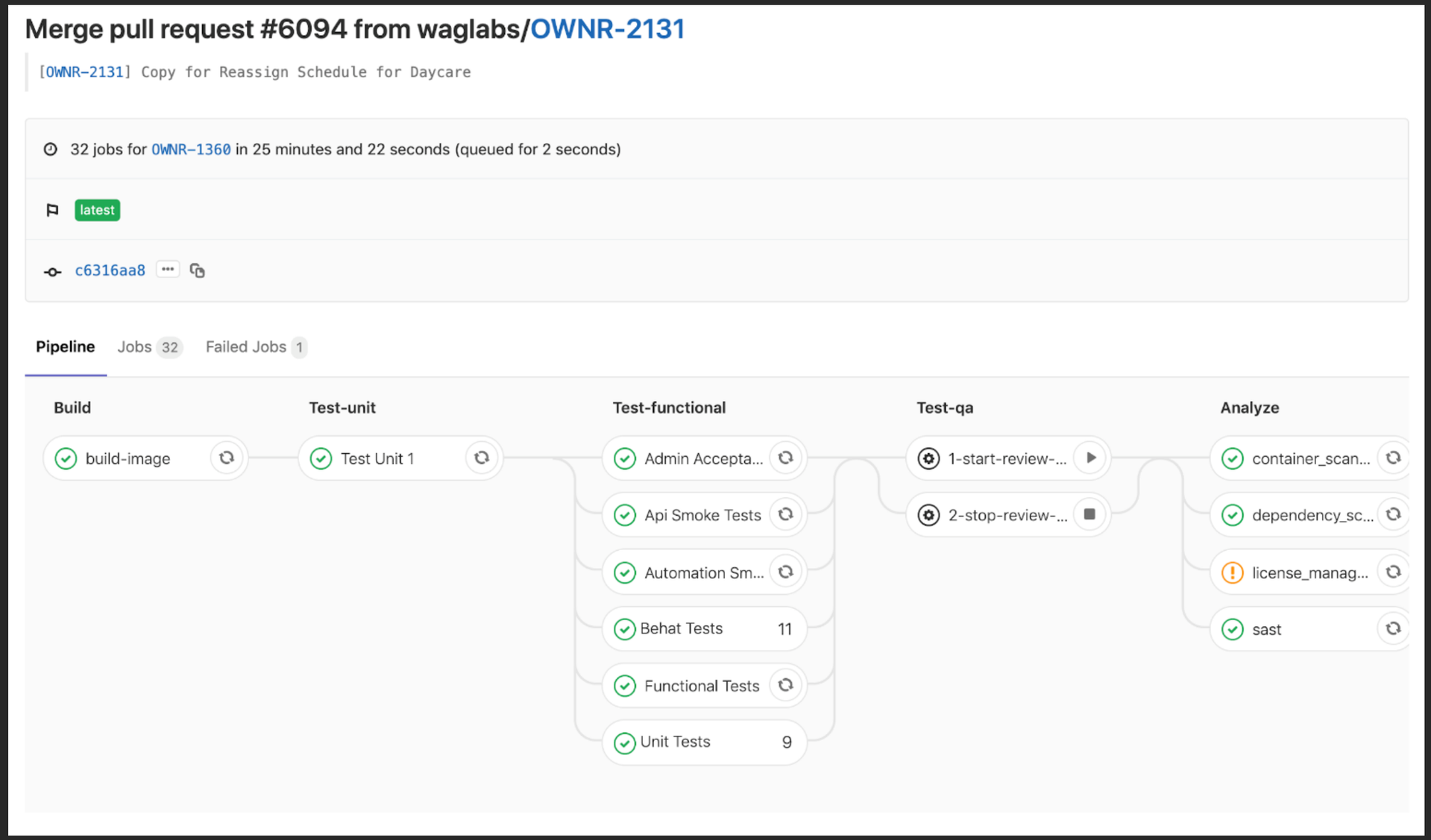
Blue/(green) architecture



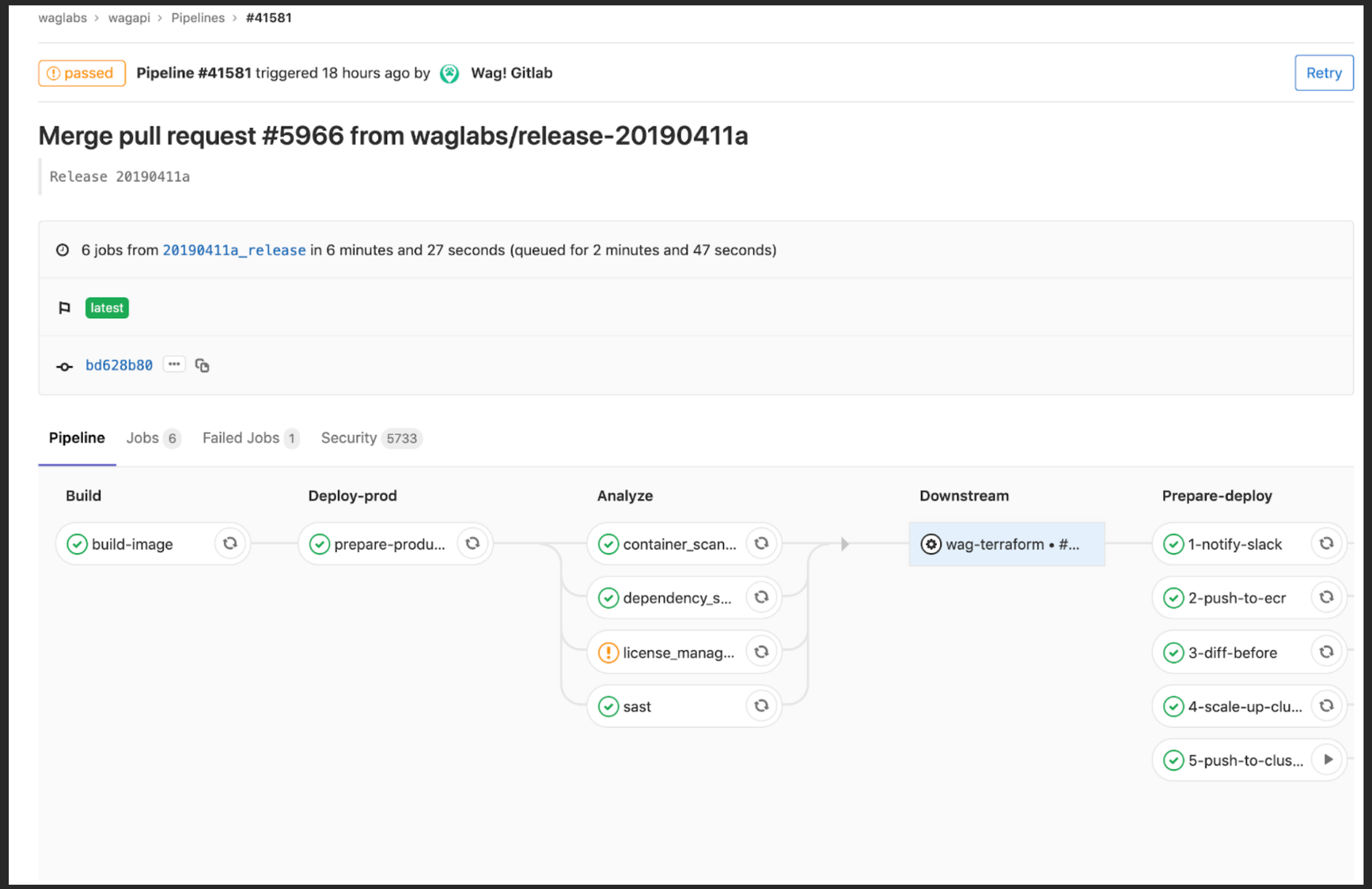
(Blue)/green architecture



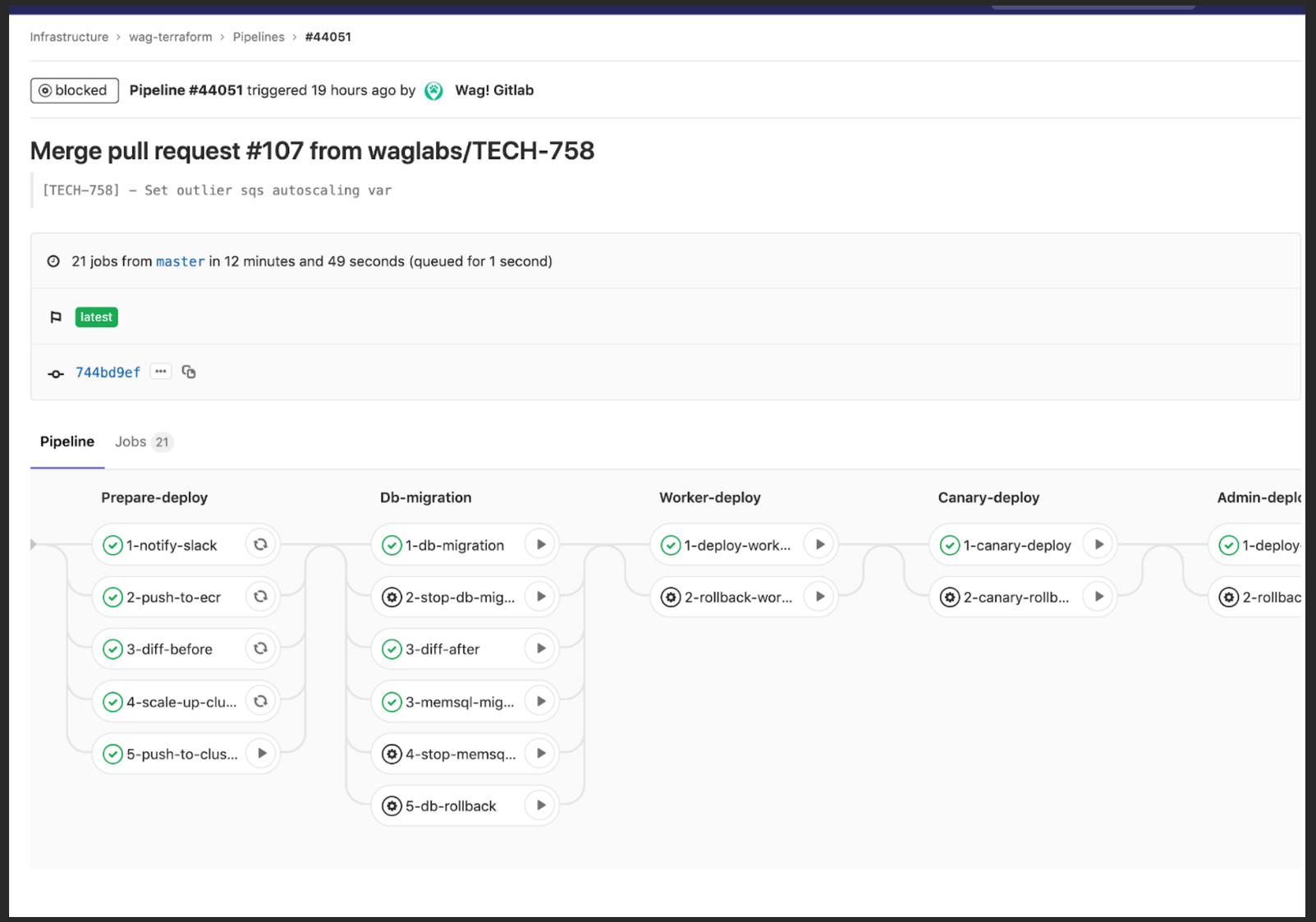
CI pipeline


Merge pull request #6094 from waglabs/OWNR-2131
[OWNR-2131] Copy for Reassign Schedule for Daycare
32 jobs for OWNR-1360 in 25 minutes and 22 seconds (queued for 2 seconds)
latest
c6316aa8
Pipeline
Jobs 32
Failed Jobs 1
Build
Test-unit
Test-functional
Test-qa
Analyze
build-image
Test Unit 1
Admin Accepta...
Api Smoke Tests
Automation Sm...
Behat Tests 11
Functional Tests
Unit Tests 9
1-start-review-...
2-stop-review-...
container_scan...
dependency_sc...
license_manag...
sast

Deployment pipeline



Deployment pipeline (continued)



ChatOps

gitlab-bot APP 2:39 PM

20190716b_release Diff Check -

```
diff  <(git diff 20190716a_release..20190716b_release) <(git diff 20190716a_release...20190716b_release)
*OK* https://github.com/waglabs/wagapi/compare/20190716a_release...20190716b_release
PR ->     3 files changed, 273 insertions(+), 2 deletions(-)
Diff ->   3 files changed, 273 insertions(+), 2 deletions(-)
/builds/infrastructure/wag-terraform/bin/wag-compare.sh release-20190716b 20190716b_release
OKAY: 0 lines of difference
```

1

gitlab APP 5:02 PM

Preparing Release

Preparing to deploy 20190716c-release to blue cluster

The current live cluster is **green** with 20190716b-release

Wag! Gitlab CI Bot

Pushing Image to ECR

Pushing 20190716c-release to blue ECR

Wag! Gitlab CI Bot

20190716c-release push to ECR complete

```
The push refers to repository [redacted]
9158216606eb: Preparing
59942c00c5de: Preparing
92d3f22d44f3: Preparing
10e46f329a25: Preparing
24ab7de5faec: Preparing
1ea5a27b0484: Preparing
1ea5a27b0484: Waiting
92d3f22d44f3: Layer already exists
24ab7de5faec: Layer already exists
10e46f329a25: Layer already exists
59942c00c5de: Layer already exists
1ea5a27b0484: Layer already exists
9158216606eb: Pushed
20190716c-release: digest: sha256:cacd205049164cb87990e3117ed53751bea49eaed1a44e1517c1e740fbe89086 size: 1576
```

20190716c-release - Diff Before (on 20190716b-release)

20190716c-release - Scaling up blue Cluster

20190716c-release - Cluster scaled up

gitlab APP 5:11 PM

Push To Inactive Cluster

Pushing 20190716c-release to blue cluster

Gitlab is deploying 20190716c-release to the inactive cluster: blue. You can check to see if the tag is active by clicking here: https://stage-blue-api-prod.wag-[redacted]

Wag! Gitlab CI Bot

20190716c-release push to cluster complete [WAIT FOR IMAGE TO BE ACTIVE]

```
api_task_definitions = arn:aws:ecs:us-west-1:[redacted]:410
release_tag = 20190716c-release
```

20190716c-release Is Ready for Cutover!!!

Starting MemSQL Migration

DB Migration

Running DB Migration

Gitlab CI is running the DB migration, output to follow:

Wag! Gitlab CI Bot

DB Migration

Running DB Migration

Gitlab CI is running the DB migration, output to follow:

Wag! Gitlab CI Bot

20190716c-release - Scaling down green Cluster

20190716c-release - Cluster scaled down

What went wrong?



© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lessons

- Change is hard
- Systems are complicated
- Things break
- Nobody is perfect
- People (me, especially) make mistakes
- ¯_(ツ)_/¯

Onward!

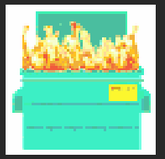


© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

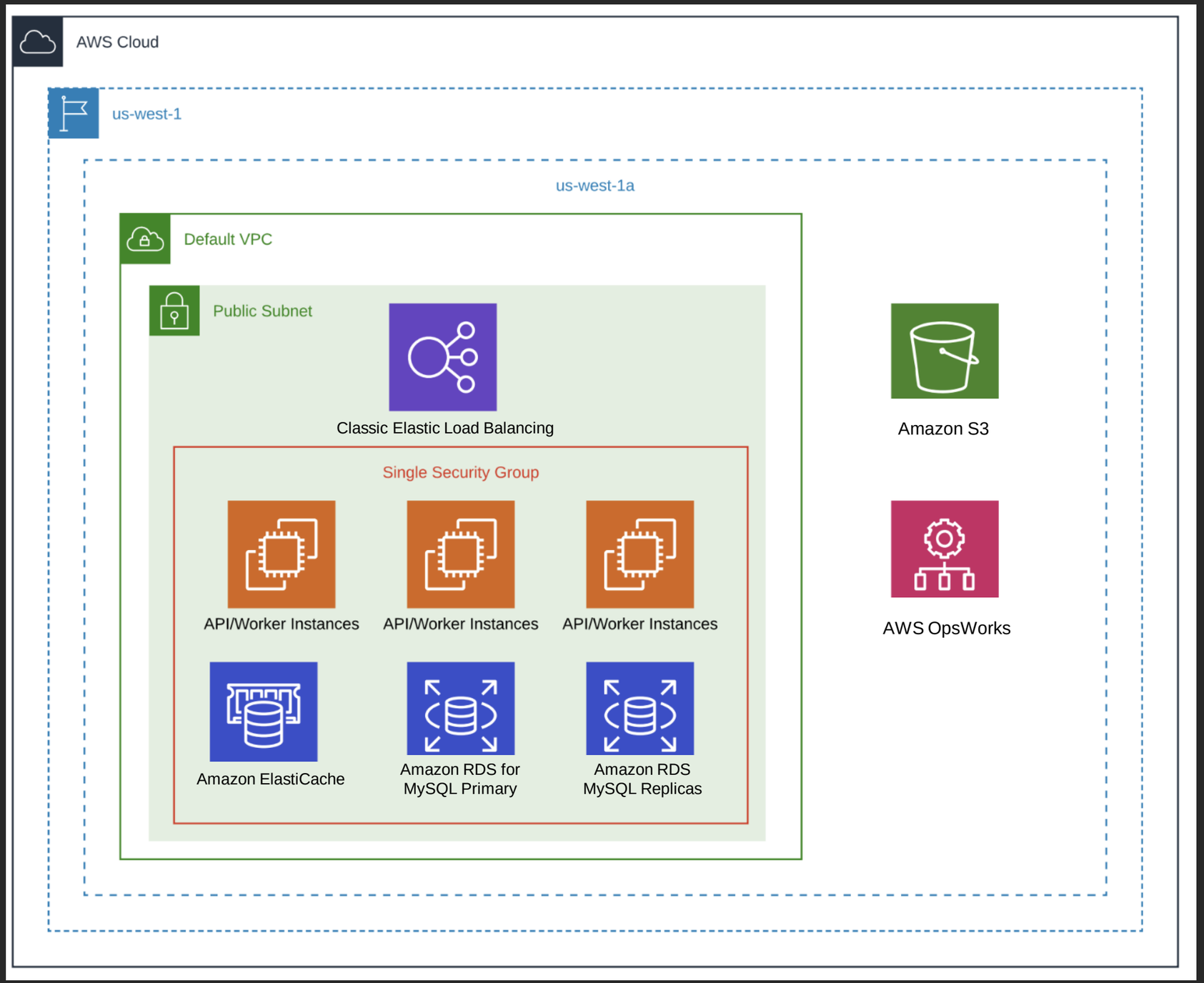
The plan: Phase 2

- Guardrails in the deployment pipeline*
- Full software development lifecycle for our infrastructure
- Split up the Terraform monorepo
- Move to Multi–AWS account Amazon S3 backend
- CI/CD for our Terraform repos with GitLab
- Build out a team

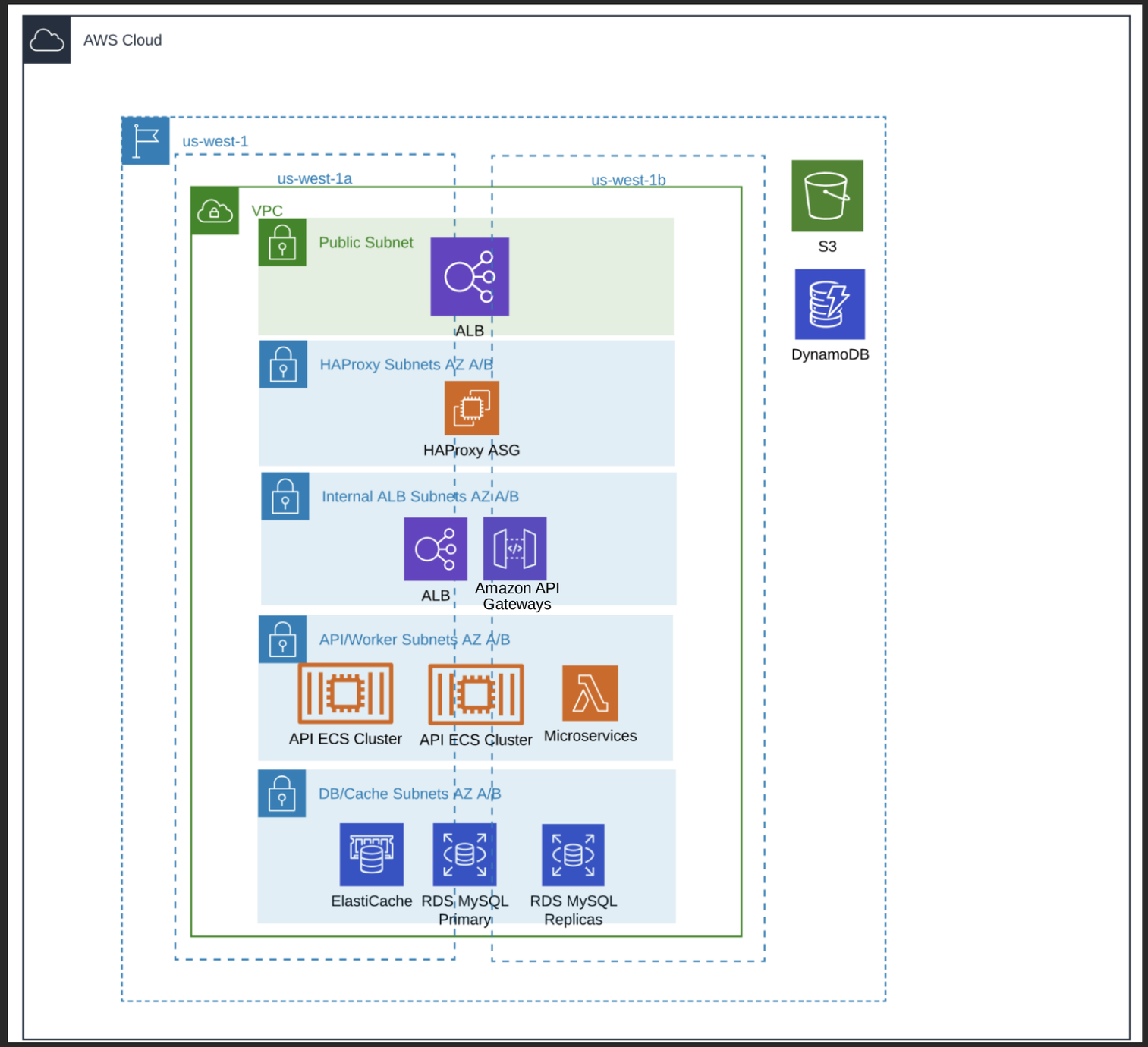
*(oops, my bad)



Reminder: What things looked like before



What things looked like after



Guardrails

- The deployment pipeline was PAW-some perfect magic
- Until it wasn’t
- If you didn’t do things in the right order, things broke 
- We added guardrails to prevent bad things
- Uptime good, downtime bad
- Guardrails!

Software development lifecycle for Terraform

We have infrastructure as code; let’s implement the software development lifecycle using GitLab!

- Cut a branch from master
- Test in your AWS sandbox
- Push your branch, lint/tests run
- Get a code review
- Merge your changes
- Automatically stage and test changes
- Review Terraform plan
- Apply plan (deploy)

Shred the monorepo

- Modules being referenced in the same repo with ../../../ wasn’t ideal
- Needed a way to lock modules via a tag
- Didn’t want a change on module to affect other environments
- Built out repos for each environment: Prod, data, Ops, QA, etc.
- Luckily, we were able to move the state as we’d structured it, similar to how Gruntwork recommends
  - live/prod/...
  - live/data/...
  - live/ops/...

Terraform multi-account Amazon S3 backend*

- Created a single administrative AWS account
  - Amazon S3 buckets for state
  - Amazon DynamoDB for locking
  - IAM users/roles for deploying in sandboxes, staging, production
- Uses workspaces and role assumption to share Terraform across multiple environments
- Makes testing, staging, and deploying much easier and scalable

*https://www.terraform.io/docs/backends/types/s3.html#multi-account-aws-architecture

CI/CD for Terraform

Now that we're following the software development lifecycle for Terraform...

- Let's PAW-tomate all the things!
- Build out pipelines for running tests, staging, planning, and applying changes in GitLab
- Push your code, get pinged in slack with a Terraform plan
- Click to apply

Build a team

- Finding good engineers is hard
- Finding good engineers who know Terraform is hard
- Hire people who are passionate
- Hire people who care about the mission and the work (and dogs, obviously)
- Hire people who are excited to learn and grow
- Hire people who like bad puns/dog puns 

A few months later

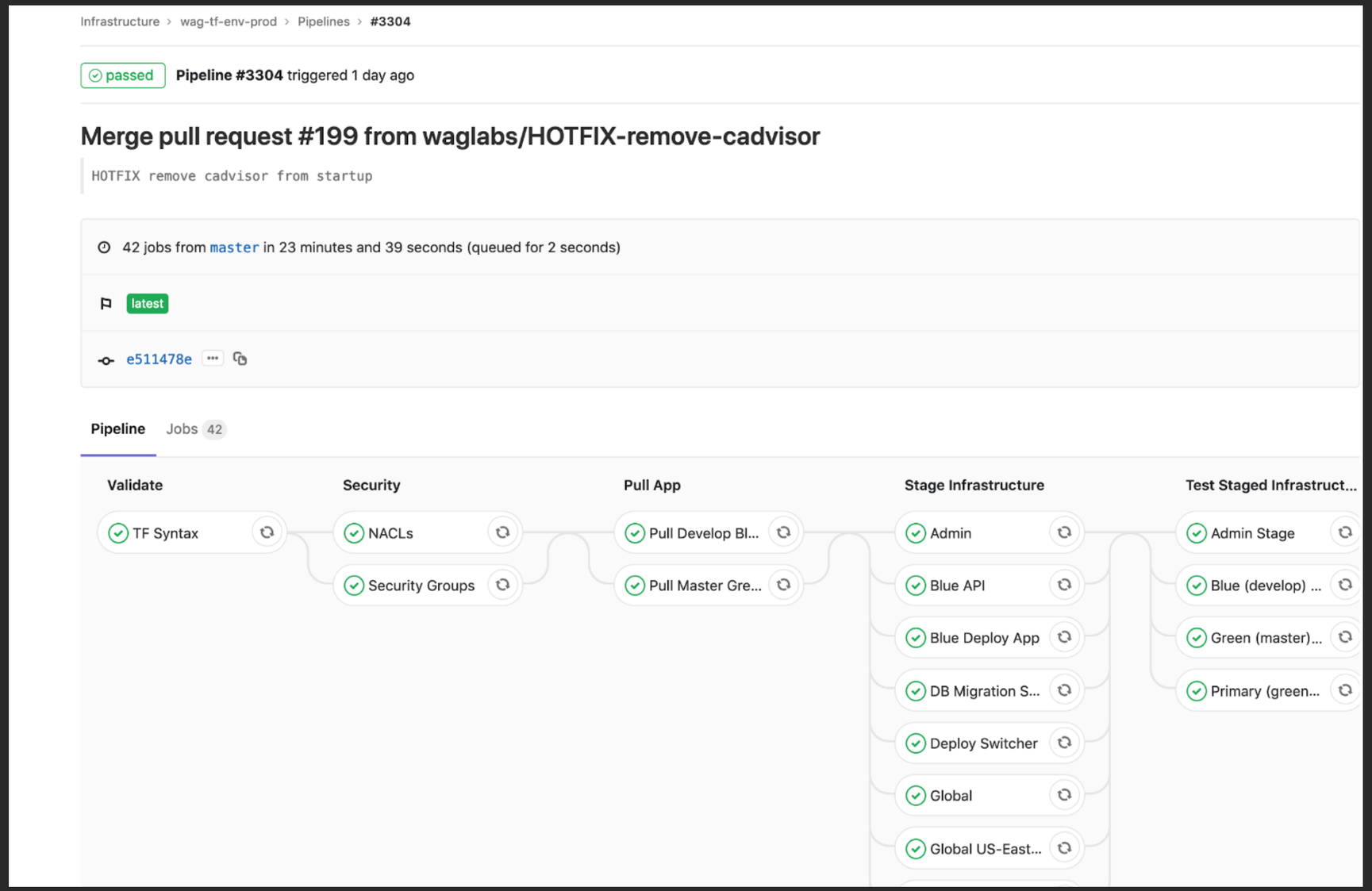
© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Success*

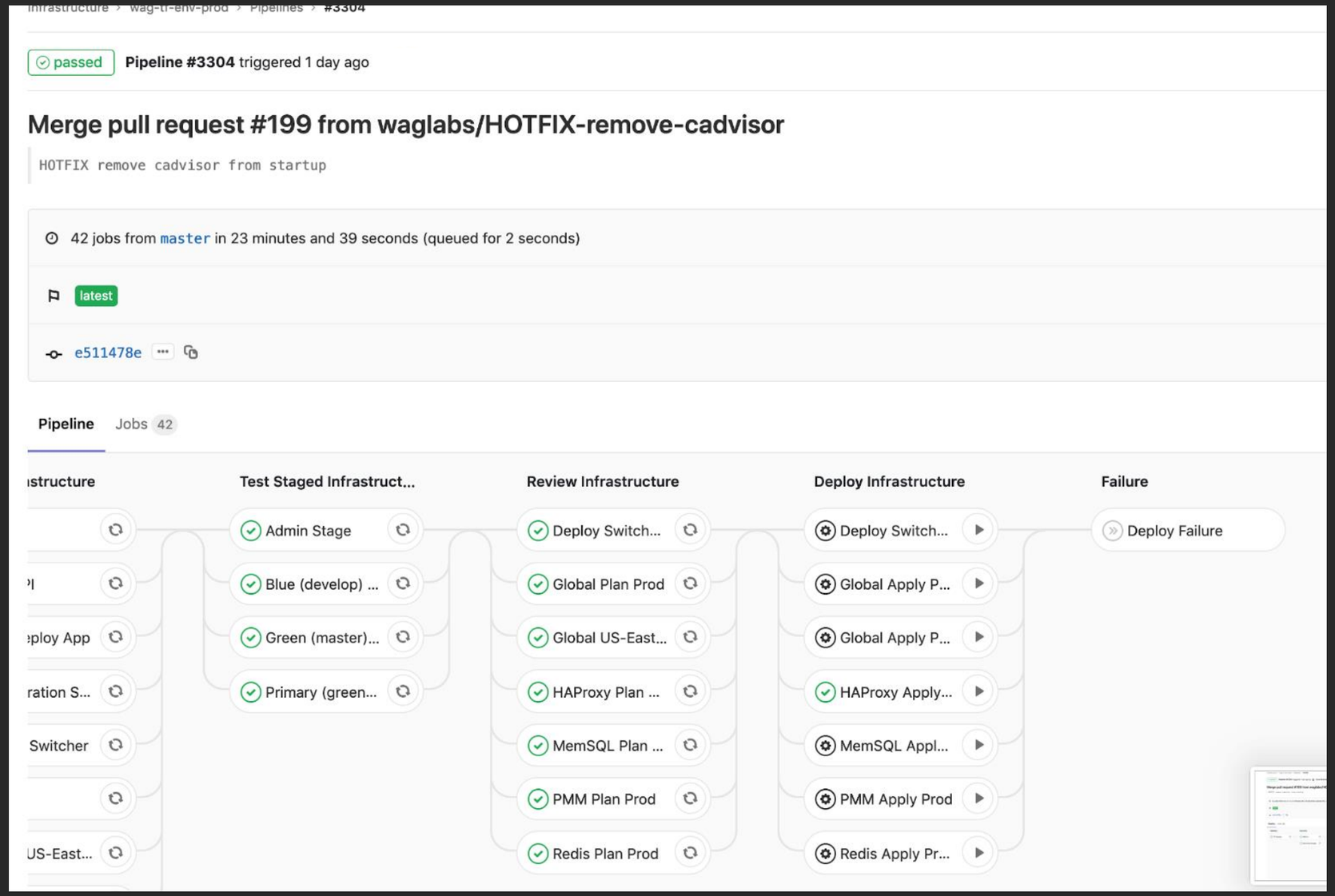
- Split out the monorepo
- Created a full CI/CD pipeline for our infrastructure
- Built out an amazing team (DogOps)
- Onboarded said team, and they didn’t all rage-quit
- Came up with so many dog puns

*Not everything was sunshine and rainbows

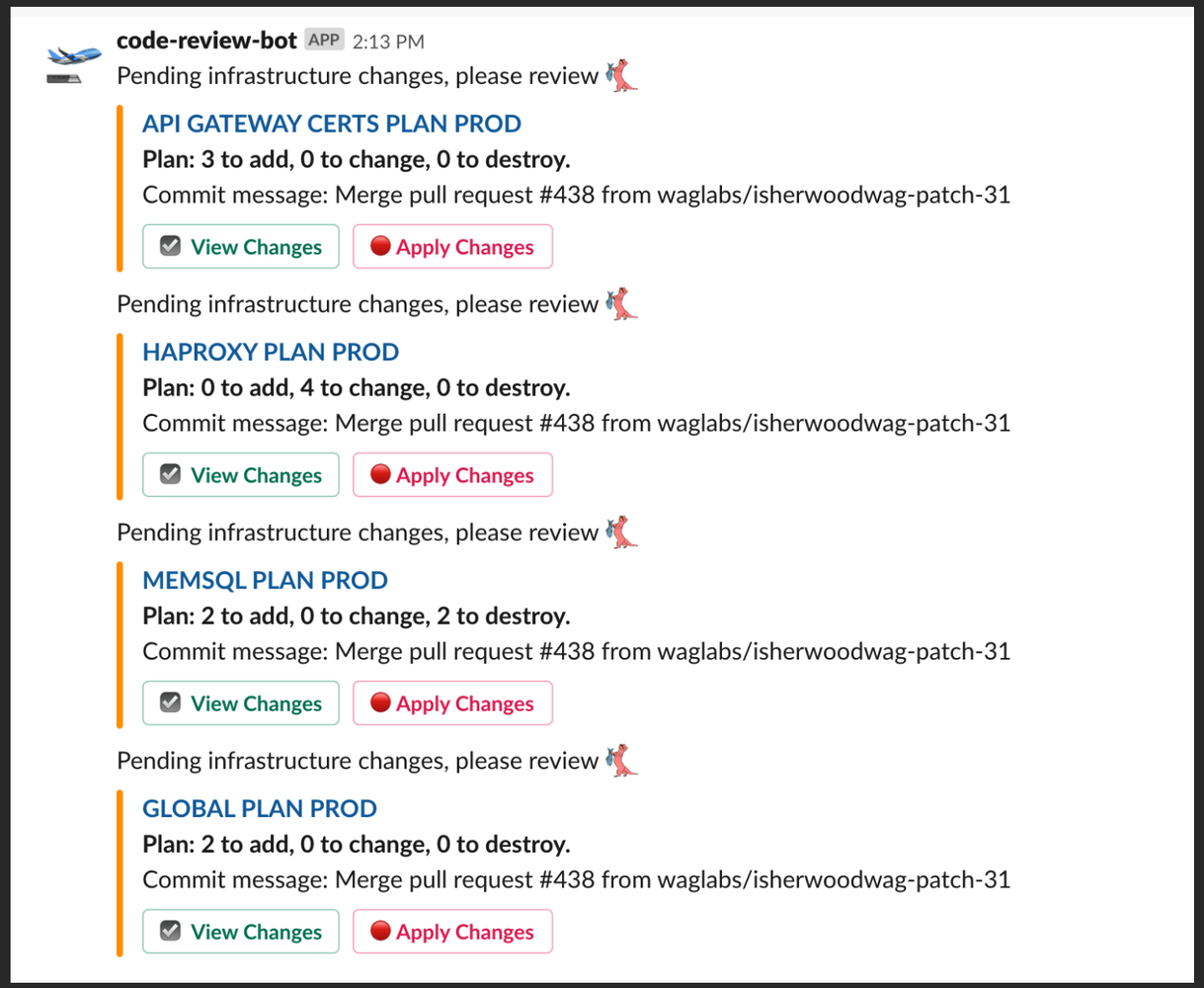
GitLab Terraform pipeline

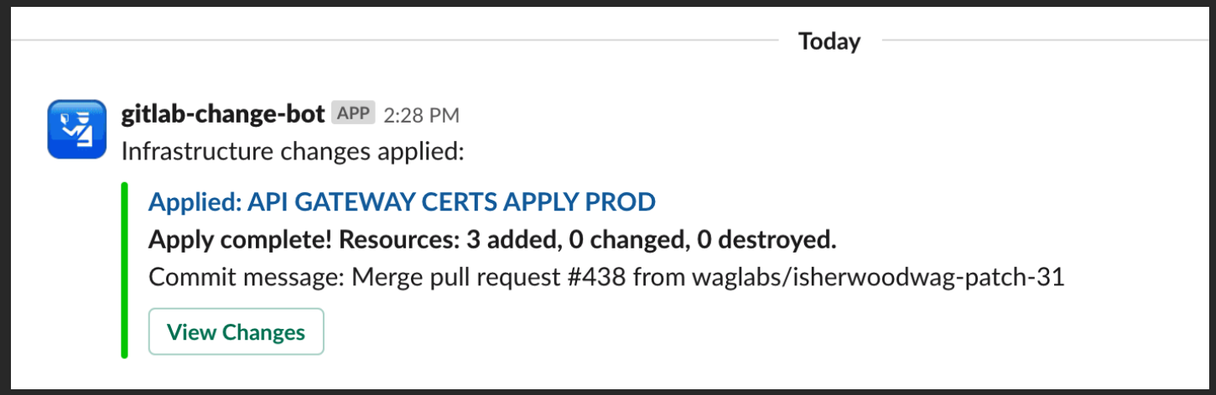


GitLab Terraform pipeline (continued)



ChatOps


code-review-bot APP 2:13 PM
Pending infrastructure changes, please review
API GATEWAY CERTS PLAN PROD
Plan: 3 to add, 0 to change, 0 to destroy.
Commit message: Merge pull request #438 from waglabs/isherwoodwag-patch-31
View Changes
Apply Changes
Pending infrastructure changes, please review
HAPROXY PLAN PROD
Plan: 0 to add, 4 to change, 0 to destroy.
Commit message: Merge pull request #438 from waglabs/isherwoodwag-patch-31
View Changes
Apply Changes
Pending infrastructure changes, please review
MEMSQL PLAN PROD
Plan: 2 to add, 0 to change, 2 to destroy.
Commit message: Merge pull request #438 from waglabs/isherwoodwag-patch-31
View Changes
Apply Changes
Pending infrastructure changes, please review
GLOBAL PLAN PROD
Plan: 2 to add, 0 to change, 0 to destroy.
Commit message: Merge pull request #438 from waglabs/isherwoodwag-patch-31
View Changes
Apply Changes


Today
gitlab-change-bot APP 2:28 PM
Infrastructure changes applied:
Applied: API GATEWAY CERTS APPLY PROD
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
Commit message: Merge pull request #438 from waglabs/isherwoodwag-patch-31
View Changes

What went wrong?



© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lessons

- Change is hard
- Systems are complicated
- Things break
- Nobody is perfect
- People (me, especially) make mistakes
- ¯_(ツ)_/¯

Onward!



© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Dave Bullock 

dave@wagwalking.com

AWS re:Invent

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws



Please complete the session survey in the mobile app.

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.