Isn’t always good for you

Who am I?

Heather Osborn

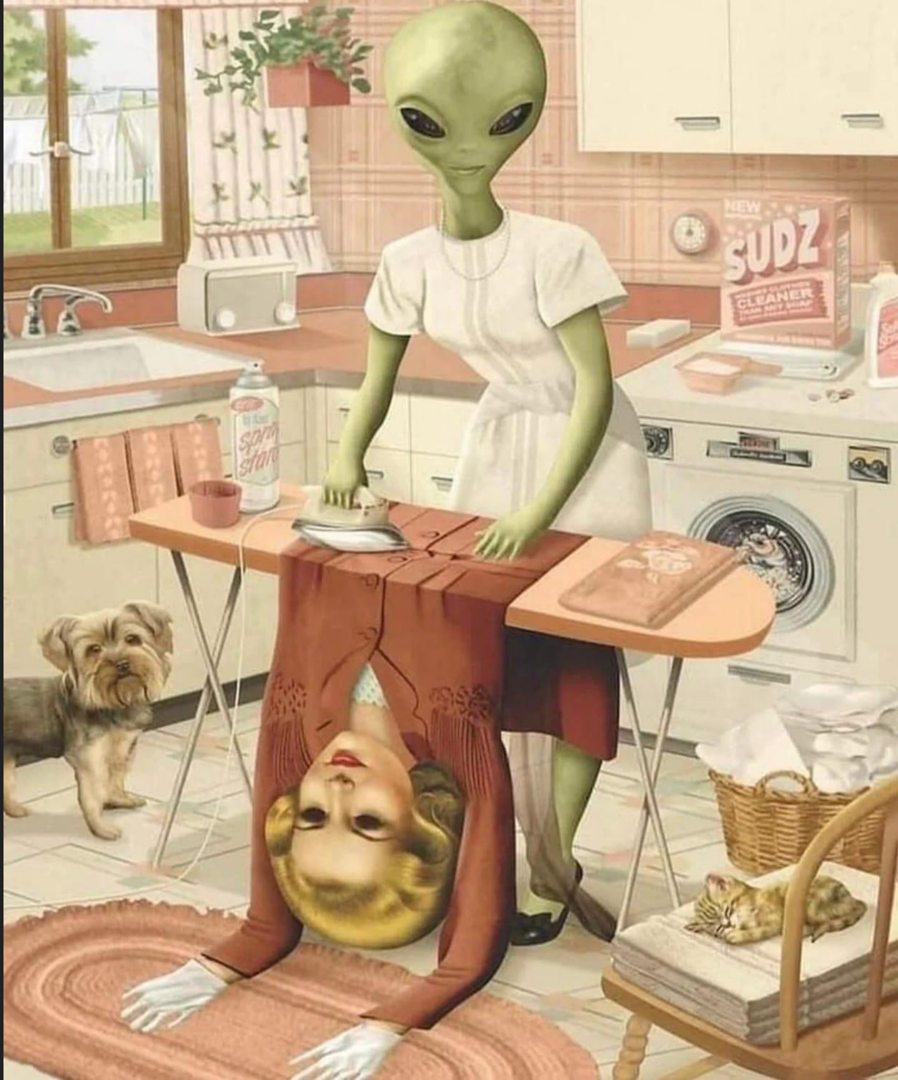
25+ years in system engineering, devops and management.

- Private cloud/on prem
- Public cloud

Crazy cat lady

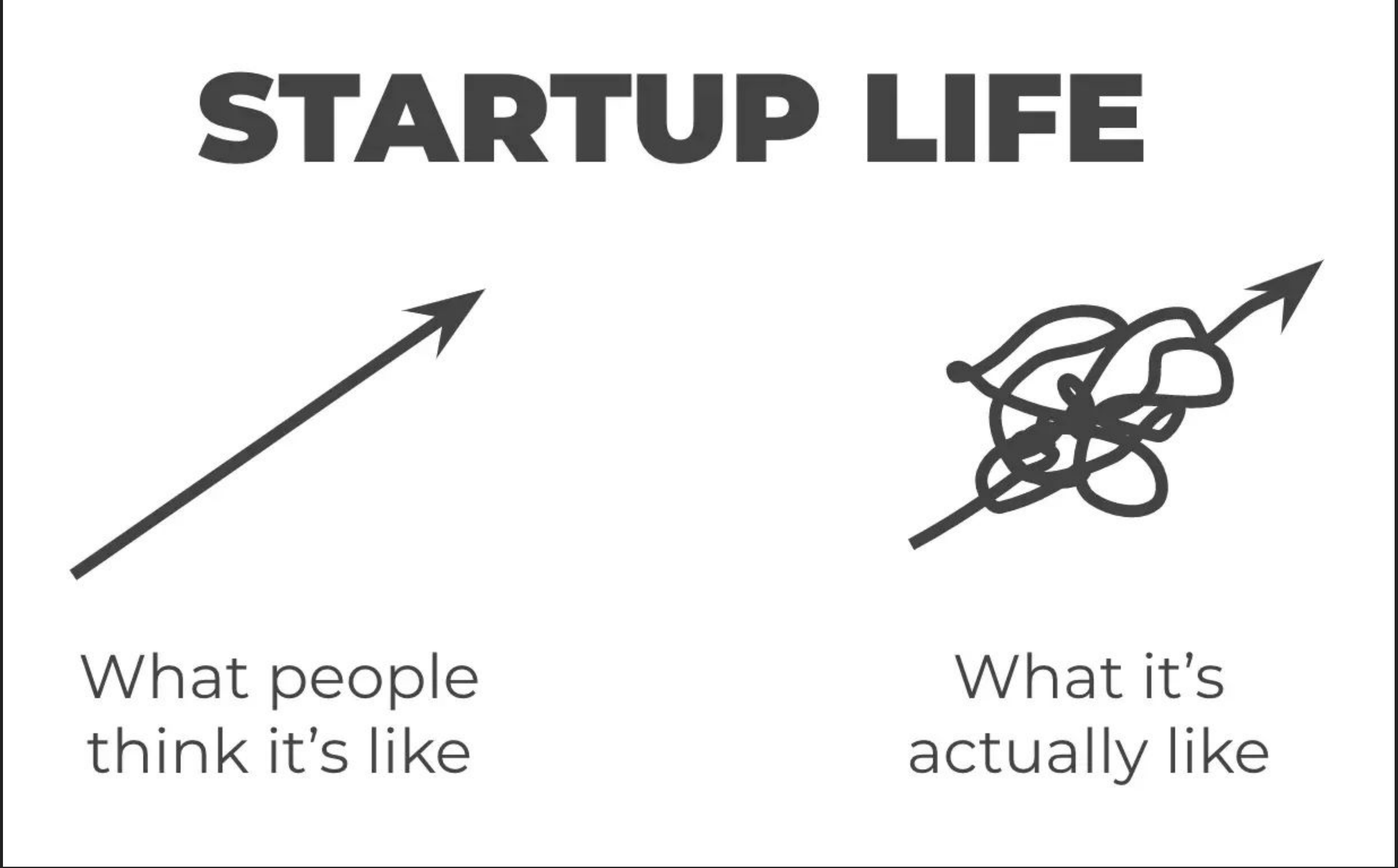
Distance runner

Immersive camping enthusiast

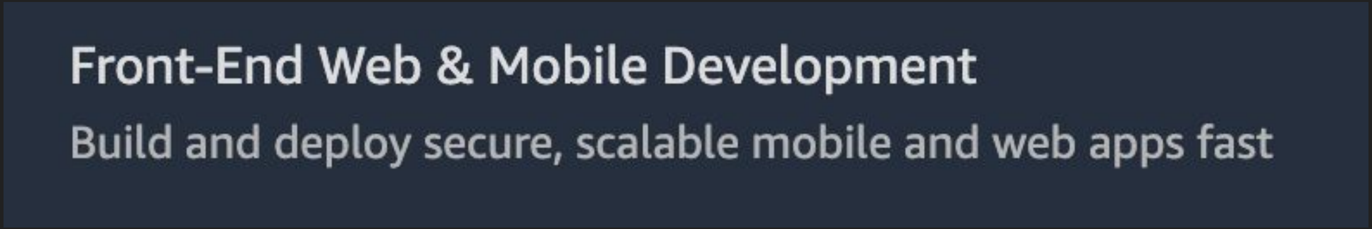


What’s the story?

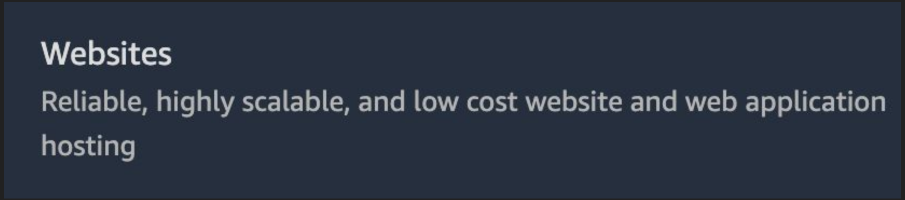
- Creating an organic mess
- Background spaghetti
- Do you really know what’s wrong?
- Woah, that’s a big problem
- Yeet the whole thing
- Tools for fun and profit


STARTUP LIFE
What people think it's like
What it's actually like

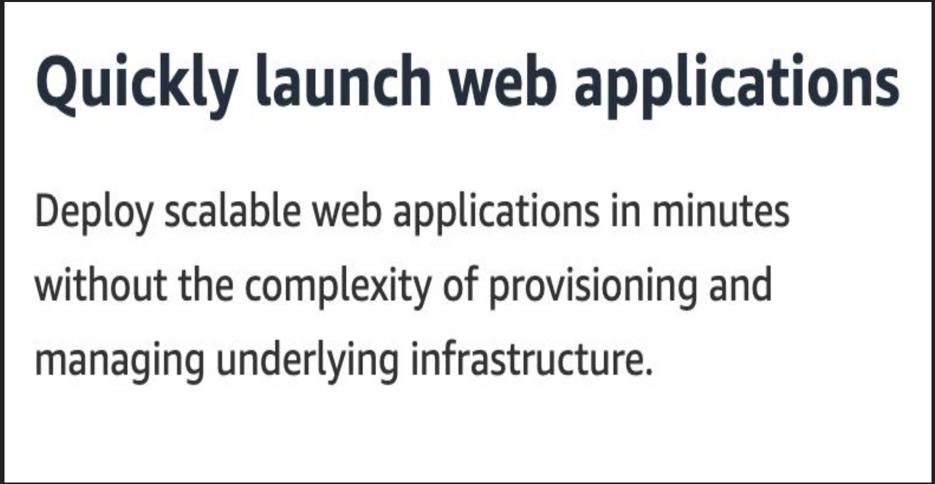
Set up an application in a day!


Front-End Web & Mobile Development

Build and deploy secure, scalable mobile and web apps fast


Websites

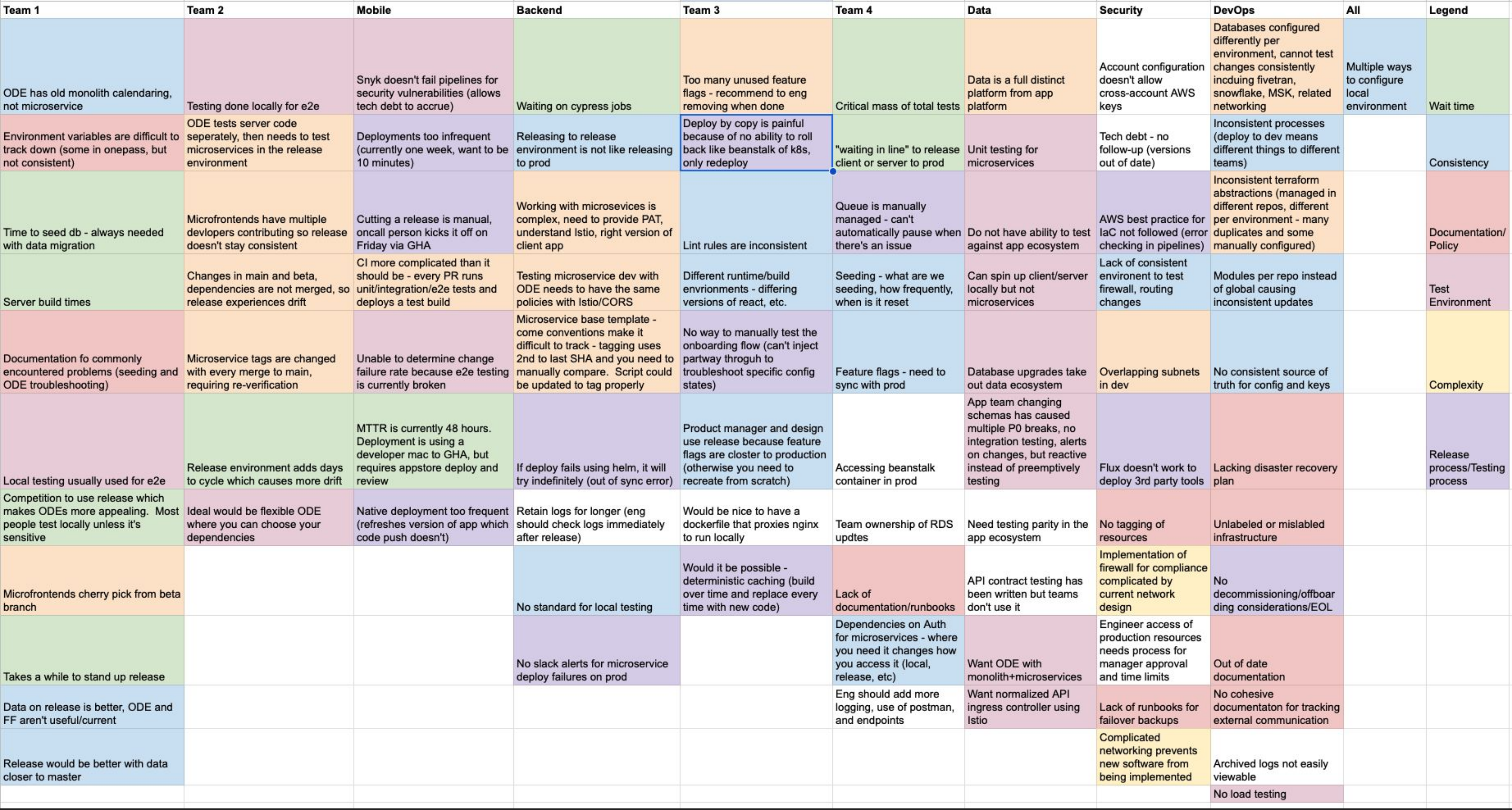
Reliable, highly scalable, and low cost website and web application hosting


Quickly launch web applications

Deploy scalable web applications in minutes without the complexity of provisioning and managing underlying infrastructure.

Background - Organic Spaghetti is especially bad for you.

Let’s figure out what the problems are…



Team 1	Team 2	Mobile	Backend	Team 3	Team 4	Data	Security	DevOps	All	Legend
ODE has old monolith calendaring, not microservice	Testing done locally for e2e	Snyk doesn't fail pipelines for security vulnerabilities (allows tech debt to accrue)	Waiting on cypress jobs	Too many unused feature flags - recommend to eng removing when done	Critical mass of total tests	Data is a full distinct platform from app platform	Account configuration doesn't allow cross-account AWS keys	Databases configured differently per environment, cannot test changes consistently incduing fivetran, snowflake, MSK, related networking	Multiple ways to configure local environment	Wait time
Environment variables are difficult to track down (some in onepass, but not consistent)	ODE tests server code seperately, then needs to test microservices in the release environment	Deployments too infrequent (currently one week, want to be 10 minutes)	Releasing to release environment is not like releasing to prod	Deploy by copy is painful because of no ability to roll back like beanstalk of k8s, only redeploy	"waiting in line" to release client or server to prod	Unit testing for microservices	Tech debt - no follow-up (versions out of date)	Inconsistent processes (deploy to dev means different things to different teams)		Consistency
Time to seed db - always needed with data migration	Microfrontends have multiple devlopers contributing so release doesn't stay consistent	Cutting a release is manual, oncall person kicks it off on Friday via GHA	Working with microsevices is complex, need to provide PAT, understand Istio, right version of client app	Lint rules are inconsistent	Queue is manually managed - can't automatically pause when there's an issue	Do not have ability to test against app ecosystem	AWS best practice for IaC not followed (error checking in pipelines)	Inconsistent terraform abstractions (managed in different repos, different per environment - many duplicates and some manually configured)		Documentation/ Policy
Server build times	Changes in main and beta, dependencies are not merged, so release experiences drift	CI more complicated than it should be - every PR runs unit/integration/e2e tests and deploys a test build	Testing microservice dev with ODE needs to have the same policies with Istio/CORS	Different runtime/build envrionments - differing versions of react, etc.	Seeding - what are we seeding, how frequently, when is it reset	Can spin up client/server locally but not microservices	Lack of consistent environent to test firewall, routing changes	Modules per repo instead of global causing inconsistent updates		Test Environment
Documentation fo commonly encountered problems (seeding and ODE troubleshooting)	Microservice tags are changed with every merge to main, requiring re-verification	Unable to determine change failure rate because e2e testing is currently broken	Microservice base template - come conventions make it difficult to track - tagging uses 2nd to last SHA and you need to manually compare. Script could be updated to tag properly	No way to manually test the onboarding flow (can't inject partway throguh to troubleshoot specific config states)	Feature flags - need to sync with prod	Database upgrades take out data ecosystem	Overlapping subnets in dev	No consistent source of truth for config and keys		Complexity
Local testing usually used for e2e	Release environment adds days to cycle which causes more drift	MTTR is currently 48 hours. Deployment is using a developer mac to GHA, but requires appstore deploy and review	If deploy fails using helm, it will try indefinitely (out of sync error)	Product manager and design use release because feature flags are closter to production (otherwise you need to recreate from scratch)	Accessing beanstalk container in prod	App team changing schemas has caused multiple P0 breaks, no integration testing, alerts on changes, but reactive instead of preemptively testing	Flux doesn't work to deploy 3rd party tools	Lacking disaster recovery plan		Release process/Testing process
Competition to use release which makes ODEs more appealing. Most people test locally unless it's sensitive	Ideal would be flexible ODE where you can choose your dependencies	Native deployment too frequent (refreshes version of app which code push doesn't)	Retain logs for longer (eng should check logs immediately after release)	Would be nice to have a dockerfile that proxies nginx to run locally	Team ownership of RDS updtes	Need testing parity in the app ecosystem	No tagging of resources	Unlabeled or mislabled infrastructure		
Microfrontends cherry pick from beta branch			No standard for local testing	Would it be possible - deterministic caching (build over time and replace every time with new code)	Lack of documentation/runbooks	API contract testing has been written but teams don't use it	Implementation of firewall for compliance complicated by current network design	No decommissioning/offboarding considerations/EOL		
Takes a while to stand up release			No slack alerts for microservice deploy failures on prod		Dependencies on Auth for microservices - where you need it changes how you access it (local, release, etc)	Want ODE with monolith+microservices	Engineer access of production resources needs process for manager approval and time limits	Out of date documentation		
Data on release is better, ODE and FF aren't useful/current					Eng should add more logging, use of postman, and endpoints	Want normalized API ingress controller using Istio	Lack of runbooks for failover backups	No cohesive documentaton for tracking external communication		
Release would be better with data closer to master							Complicated networking prevents new software from being implemented	Archived logs not easily viewable		
								No load testing		

What was DevOps going to do?

Choice 1: Greenfield?

Choice 2: Upgrade/clean up prod, copy to dev for parity, update prod, repeat

Choice 3: Improve in place

Greenfield Go Ahead

What do we want?

- Never ever do anything that's not in code
- Validate validate validate
- Keep documentation close to code
- Simplify product engineer life
- Testability
- Consistent environments
- Low touch release process with visibility
    - Consistent promotion process
- NO SPAGHETTI

It’s so clean…

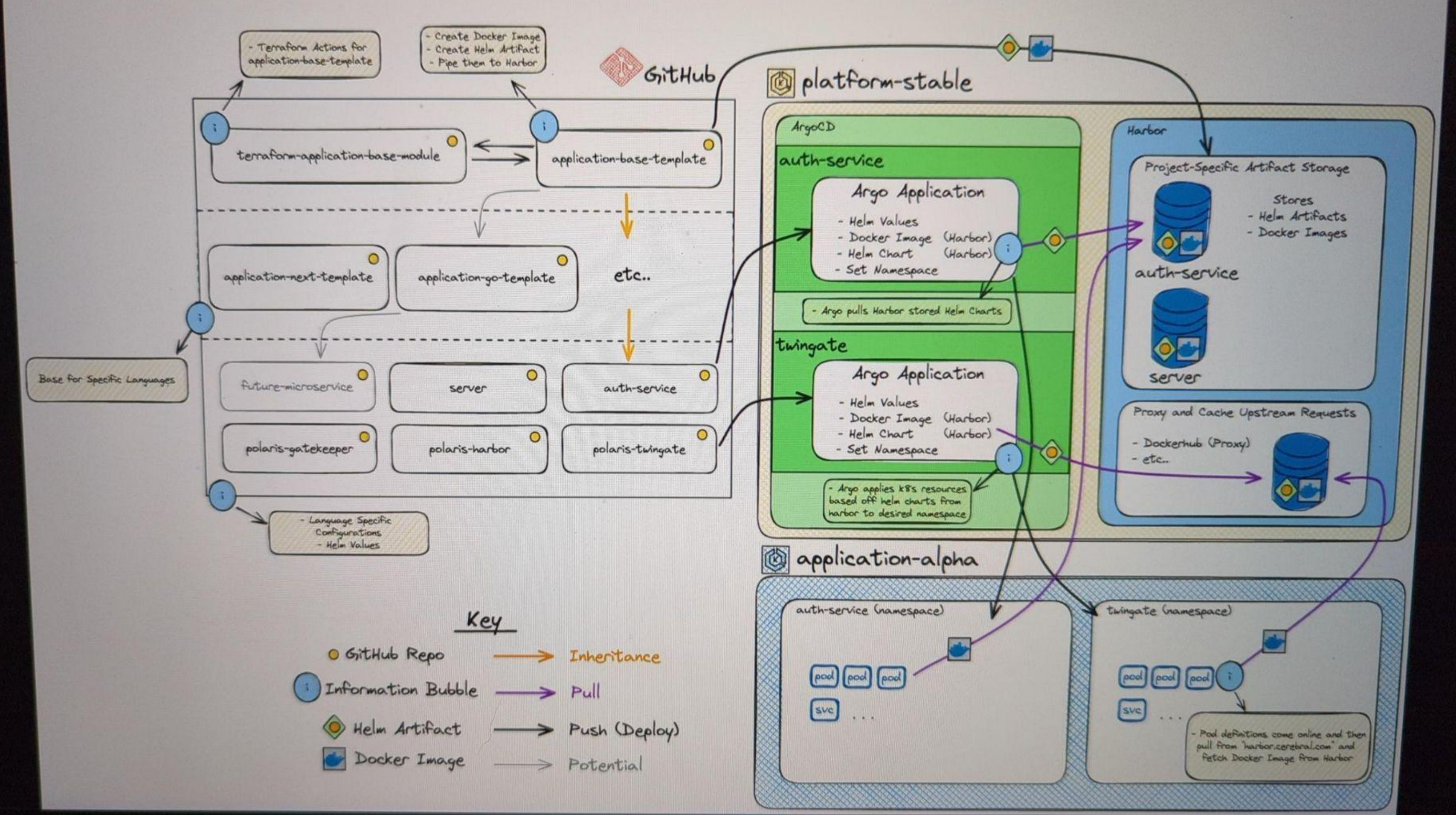
Platform cluster (tools)

- Alpha
- Beta
- Stable

App cluster

- Alpha
- Beta
- Stable

Visualize Consumption of Application Templates



Validate EVERYTHING

Validate in local - pre-commit hooks

Validate in CI - conftest

Deployment validation - Gatekeeper/OPA

Validate cluster runtime - GuardDuty

Tools Tools Tools

argo

Kargo

Amazon EKS

Any Questions?