Building a Cloud-Native Technology Stack that Supports Full Cycle Development

Daniel Bryant

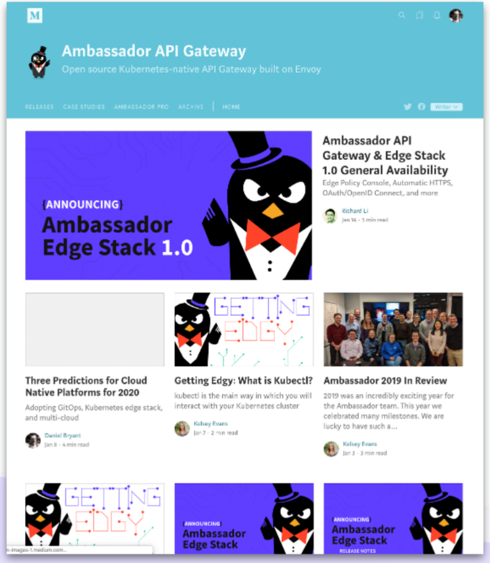
Product Architect, Ambassador Labs (formerly Datawire)

DATAWIRE

tl;dr

- Being fully cloud native requires new tech and new workflows
- Creating a supporting cloud platform is essential:
  - Container orchestration
  - Progressive delivery
  - Edge management
  - Observability
- Consciously design your platform & watch for antipatterns

@danielbryantuk



Ambassador API Gateway

Open source Kubernetes-native API Gateway built on Envoy

RELEASES CASE STUDIES AMBASSADOR PRO ARCHIVE | HOME

{ANNOUNCING} Ambassador Edge Stack 1.0

Ambassador API Gateway & Edge Stack 1.0 General Availability

Edge Policy Console, Automatic HTTPS, OAuth/OpenID Connect, and more

Richard Li
Jan 16 · 3 min read

Three Predictions for Cloud Native Platforms for 2020

Adopting GitOps, Kubernetes edge stack, and multi-cloud

Daniel Bryant
Jan 8 · 4 min read

Getting Edgy: What is Kubectl?

kubectl is the main way in which you will interact with your Kubernetes cluster

Kelsey Evans
Jan 7 · 2 min read

Ambassador 2019 In Review

2019 was an incredibly exciting year for the Ambassador team. This year we celebrated many milestones. We are lucky to have such a…

Kelsey Evans
Jan 3 · 3 min read

GETTING EDGY
{ANNOUNCING} Ambassador Edge Stack
{ANNOUNCING} Ambassador Edge Stack RELEASE NOTES

Facilitating The Spread Of Knowledge And Innovation In Professional Software Development | More

InfoQ
Development
Architecture & Design
AI, ML and Data Engineering
Culture & Methods
DevOps
NEW Videos with Transcripts
QCon
FEATURED: Streaming Machine Learning Reactive Microservices Containers Observability APM
eMag: Operationalizing Microservices
EDIT PROFILE
InfoQ Editor

Daniel Bryant

Profile page created Dec 11, 2014

Independent Tech Consultant | Consulting CTO | InfoQ Editor

Daniel Bryant blogs at https://medium.com/@danielbryantuk

Summary

Daniel Bryant is leading change within organisations and technology. His current work includes enabling agility within organisations by introducing better requirement gathering and planning techniques, focusing on the relevance of architecture within agile development, and facilitating continuous integration/delivery. Daniel's current technical expertise focuses on 'DevOps' tooling, cloud/container platforms and microservice implementations. He is also a leader within the London Java Community (LJC), contributes to several open source projects, writes for well-known technical websites such as InfoQ, DZone and Voxxed, and regularly presents at international conferences such as QCon, JavaOne and

4 Following 1232 Followers

ALL ACTIVITY

All Activity	
Articles	27
News	246
Presentations	5
Interviews	4
Podcasts	5
Reviewed	61
Likes	40
Comments	69
Following	4
Followers	1232

All Activity

Oliver Gould on the Three Bets

In this podcast, we summarise the
visibility, security, and reliability.

Reviewed this · 20 SEP, 2019 08:00

HashiConf US 2019: Terrafo

At the fifth HashiConf US conference,
Terraform and Consul products. Add
delivery world is becoming multi-clou
and vice versa.

Authored this · 19 SEP, 2019 08:00

O'REILLY

Continuous Delivery in Java

ESSENTIAL TOOLS AND BEST PRACTICES FOR DEPLOYING CODE TO PRODUCTION

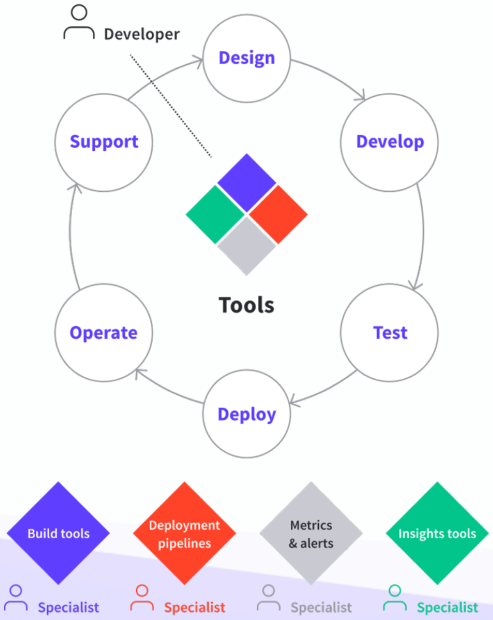
Daniel Bryant
& Abraham Marín-Pérez

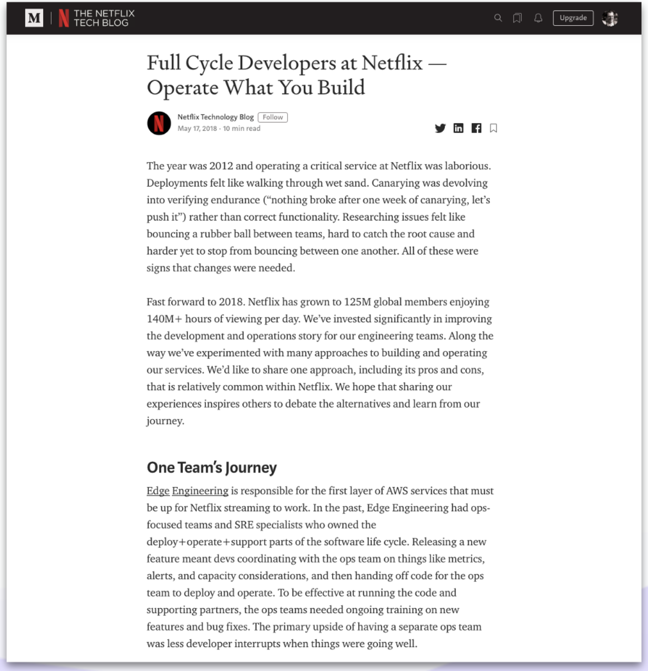
A quick cloud native primer...

- Going "cloud native" offers benefits, but requires changes:
  - New technologies
  - Appropriate culture
  - New workflows

- Successful cloud native organisations have:
  - Created a self-service application platform
  - Adopted new tools and (full cycle) developer workflows

Full Cycle Developers





Full Cycle Developers at Netflix — Operate What You Build

Netflix Technology Blog Follow
May 17, 2018 · 10 min read

The year was 2012 and operating a critical service at Netflix was laborious. Deployments felt like walking through wet sand. Canarying was devolving into verifying endurance (“nothing broke after one week of canarying, let’s push it”) rather than correct functionality. Researching issues felt like bouncing a rubber ball between teams, hard to catch the root cause and harder yet to stop from bouncing between one another. All of these were signs that changes were needed.

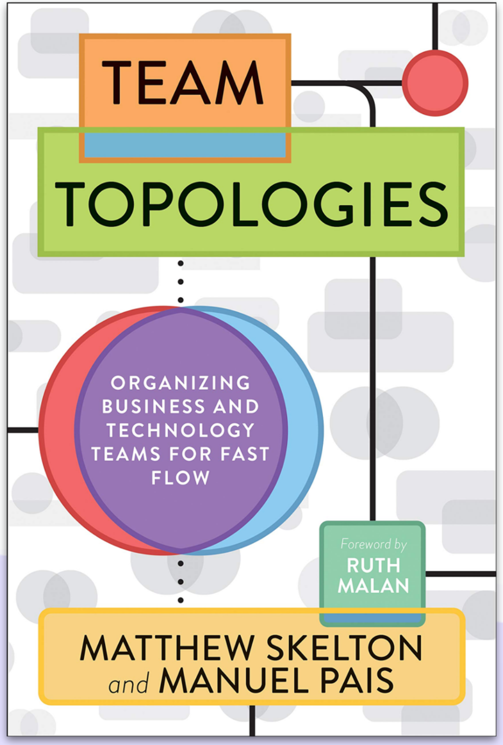
Fast forward to 2018. Netflix has grown to 125M global members enjoying 140M+ hours of viewing per day. We’ve invested significantly in improving the development and operations story for our engineering teams. Along the way we’ve experimented with many approaches to building and operating our services. We’d like to share one approach, including its pros and cons, that is relatively common within Netflix. We hope that sharing our experiences inspires others to debate the alternatives and learn from our journey.

One Team’s Journey

Edge Engineering is responsible for the first layer of AWS services that must be up for Netflix streaming to work. In the past, Edge Engineering had ops-focused teams and SRE specialists who owned the deploy+operate+support parts of the software life cycle. Releasing a new feature meant devs coordinating with the ops team on things like metrics, alerts, and capacity considerations, and then handing off code for the ops team to deploy and operate. To be effective at running the code and supporting partners, the ops teams needed ongoing training on new features and bug fixes. The primary upside of having a separate ops team was less developer interrupts when things were going well.

https://netflixtechblog.com/full-cycle-developers-at-netflix-a08c31f83249

Full Cycle Developers: Team Topologies

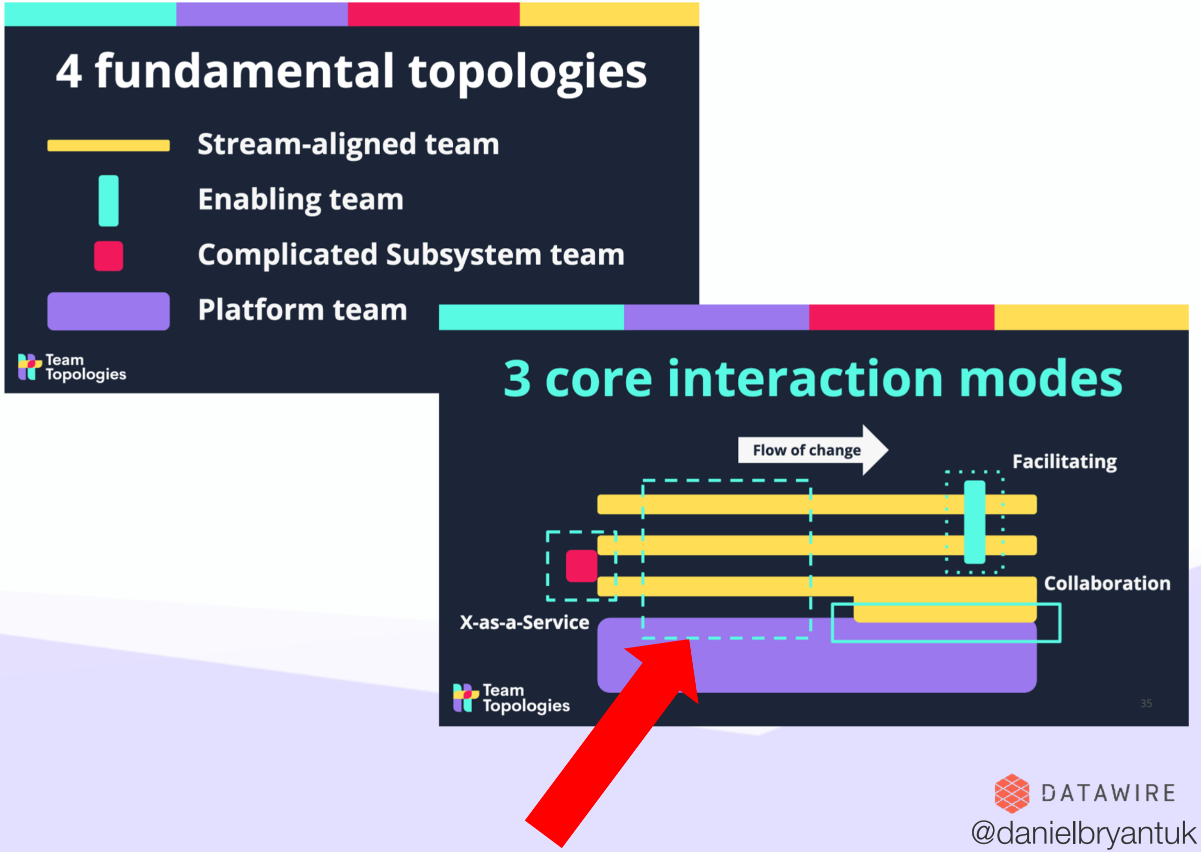


TEAM TOPOLOGIES

ORGANIZING BUSINESS AND TECHNOLOGY TEAMS FOR FAST FLOW

Foreword by RUTH MALAN

MATTHEW SKELTON and MANUEL PAIS



4 fundamental topologies

- Stream-aligned team
- Enabling team
- Complicated Subsystem team
- Platform team

Team Topologies

3 core interaction modes

Team Topologies

35

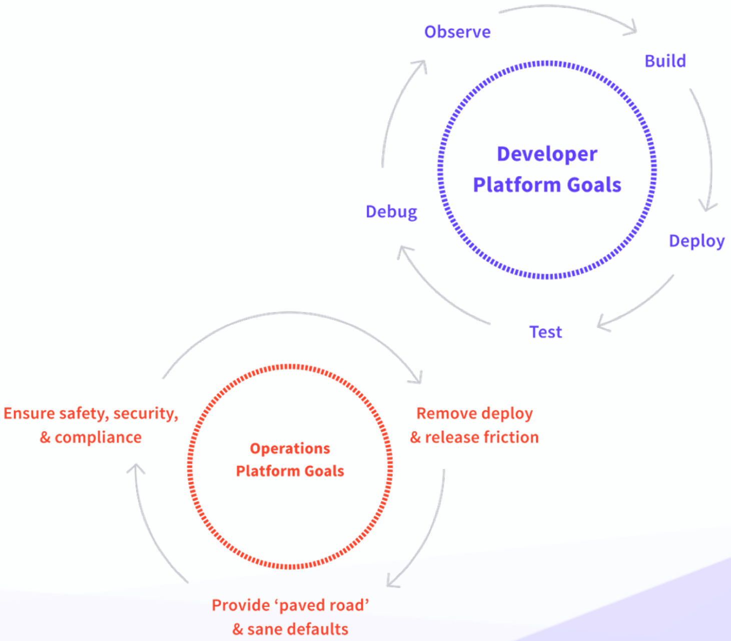
Four cloud native platform requirements

1. Container management

1. Progressive delivery

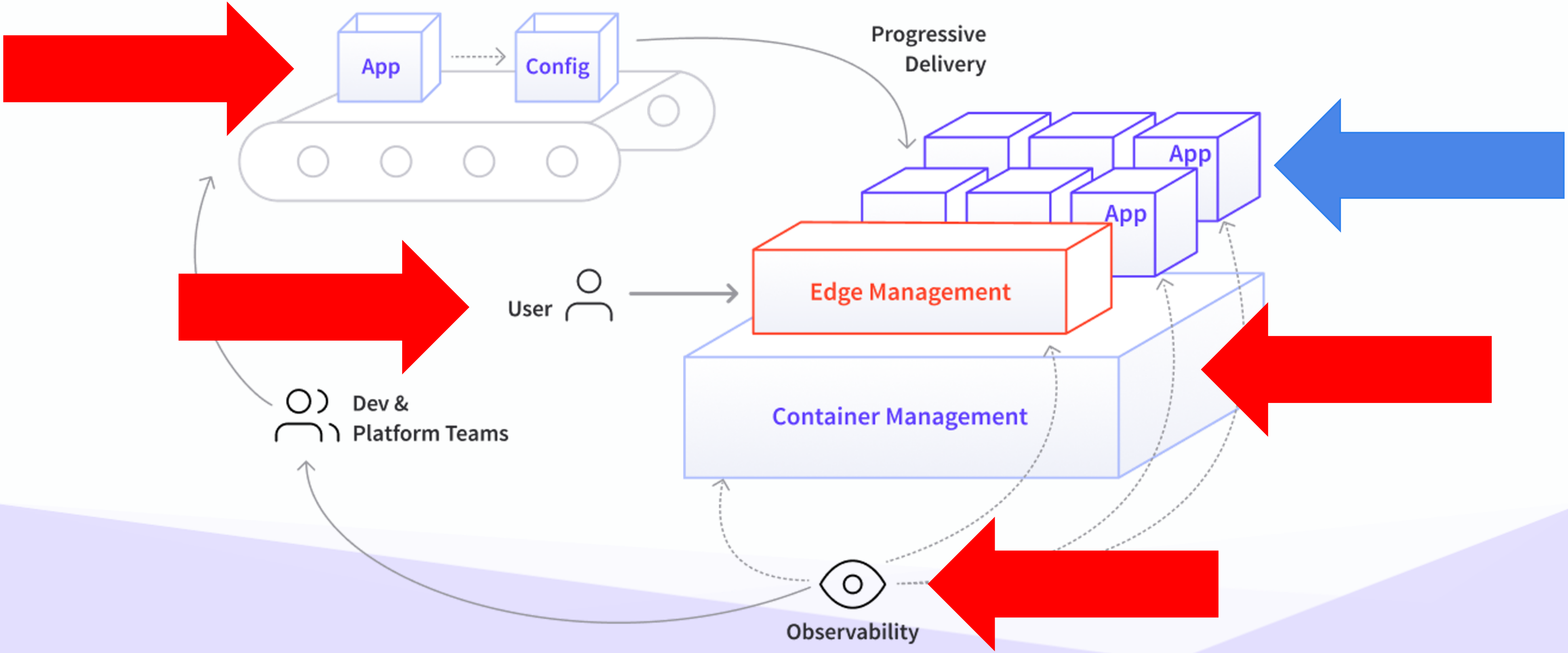
1. Edge management

1. Observability


Observe
Build
Developer Platform Goals
Debug
Deploy
Test
Ensure safety, security, & compliance
Operations Platform Goals
Remove deploy & release friction
Provide ‘paved road’ & sane defaults

DATAWIRE

@danielbryantuk



App

Config

Progressive Delivery

App

App

User

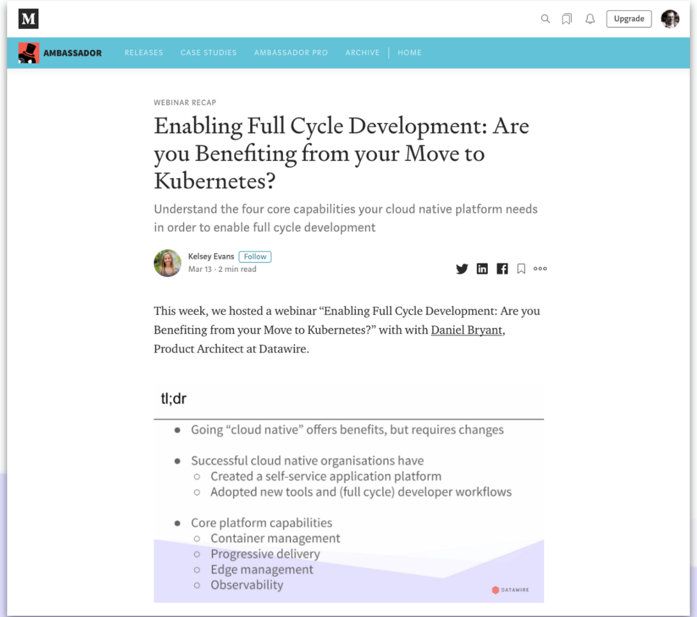
Edge Management

Container Management

Dev & Platform Teams

Observability

More Details on Full Cycle and K8s



WEBINAR RECAP

Enabling Full Cycle Development: Are you Benefiting from your Move to Kubernetes?

Understand the four core capabilities your cloud native platform needs in order to enable full cycle development

Kelsey Evans
Mar 13 · 2 min read

This week, we hosted a webinar "Enabling Full Cycle Development: Are you Benefiting from your Move to Kubernetes?" with with Daniel Bryant, Product Architect at Datawire.

tl;dr

- Going "cloud native" offers benefits, but requires changes
- Successful cloud native organisations have
  - Created a self-service application platform
  - Adopted new tools and (full cycle) developer workflows
- Core platform capabilities
  - Container management
  - Progressive delivery
  - Edge management
  - Observability

- Successful cloud native organisations have:
  - Created a self-service application platform
  - Adopted new tools and (full cycle) developer workflows

https://blog.getambassador.io/enabling-full-cycle-development-are-you-benefiting-from-your-move-to-kubernetes-d9eab2e94e7

DATAWIRE
@danielbryantuk

Avoiding Platform Antipatterns

Avoiding Platform Antipatterns

Centralized Control and Ownership: One Size Doesn’t Fit All

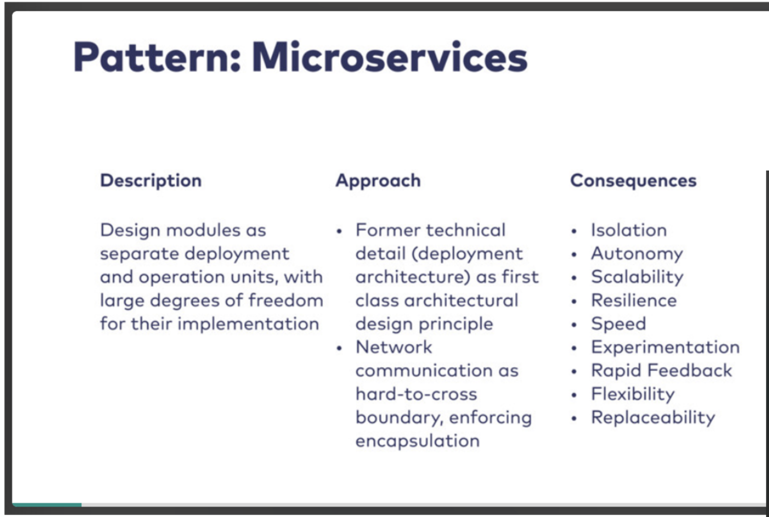
Fragmented Platform Implementation

Slow Development Loops: Less Time Coding, More Time Toiling

Antipattern: Centralized Control and Ownership

- (Dis)economies of scale
- Overzealous guardrails
- Modification is ticket-driven

Antipattern: Fragmented Platform Implementation



Pattern: Microservices

Description

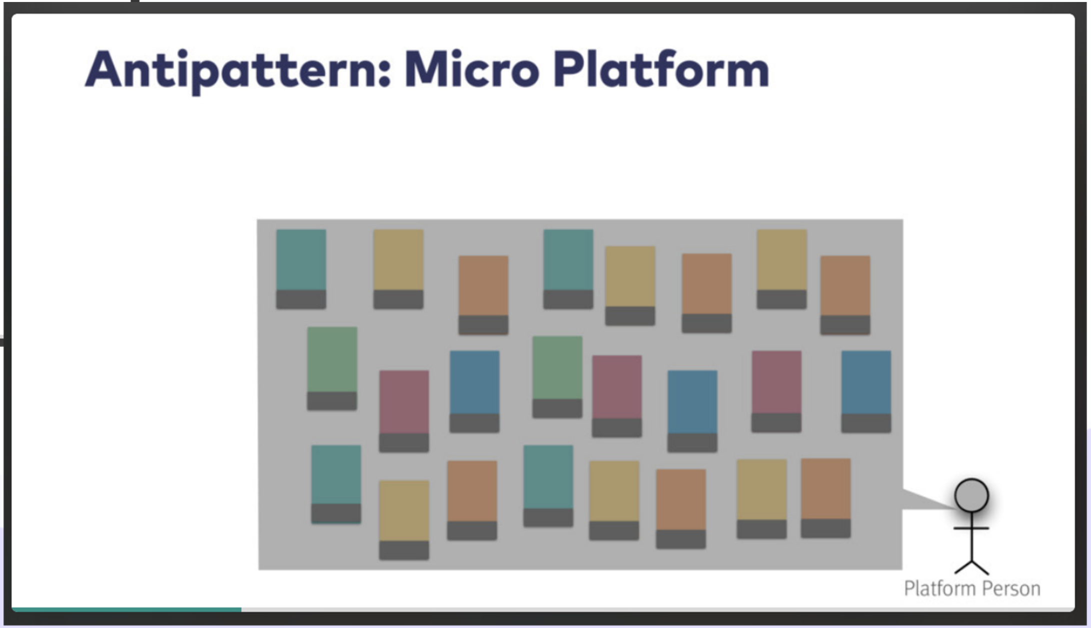
Design modules as separate deployment and operation units, with large degrees of freedom for their implementation

Approach

- Former technical detail (deployment architecture) as first class architectural design principle
- Network communication as hard-to-cross boundary, enforcing encapsulation

Consequences

- Isolation
- Autonomy
- Scalability
- Resilience
- Speed
- Experimentation
- Rapid Feedback
- Flexibility
- Replaceability

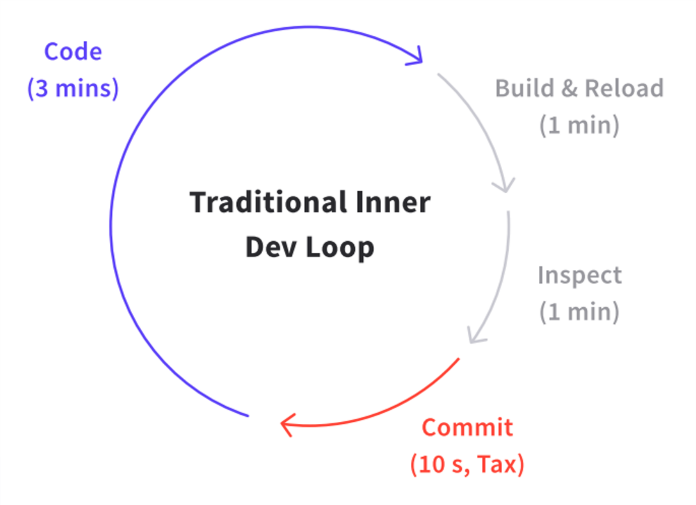


Antipattern: Micro Platform

https://speakerdeck.com/stilkov/microservices-patterns-and-antipatterns-1?slide=12

DATAWIRE

Antipattern: Slow Development Loops



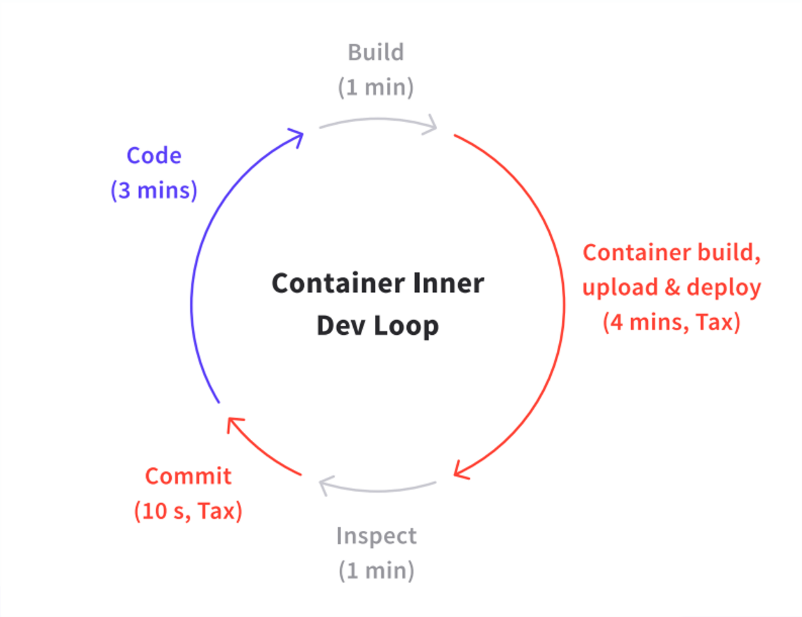
Traditional Inner Dev Loop

Code (3 mins)

Build & Reload (1 min)

Inspect (1 min)

Commit (10 s, Tax)



Container Inner Dev Loop

Build (1 min)

Code (3 mins)

Container build, upload & deploy (4 mins, Tax)

Commit (10 s, Tax)

Inspect (1 min)

https://mitchdenny.com/the-inner-loop/

Exploring the Platform Capabilities

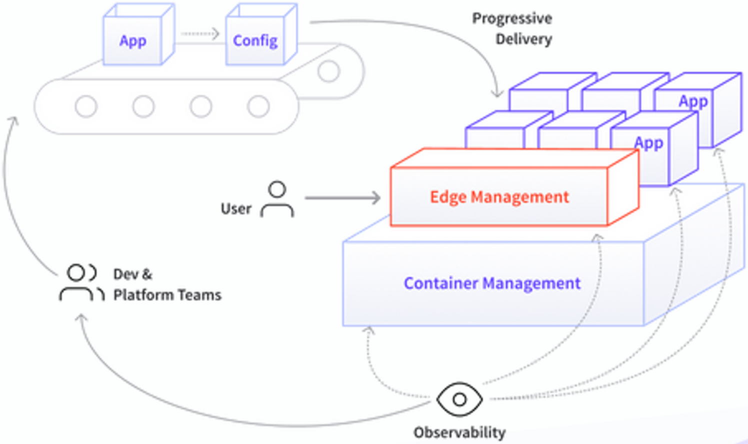
Four Core Platform Capabilities

1. Container management

1. Progressive delivery

1. Edge management

1. Observability



DATAWIRE

@danielbryantuk

Container Management: Kubernetes

Container Management

Manage and run container-based applications at scale and on a variety of infrastructures

- Developers
  - Self-service interactions: automated and observable
- Platform team
  - Set policies around access, control, and auditability

Kubernetes Decisions

- To self-host, or not to self-host?

kops

kubeadm

- Which distro?

Red Hat OpenShift

RANCHER®

- Going all-in on a cloud?

Amazon EKS

Google Kubernetes Engine

Azure Kubernetes Service (AKS)

Kubernetes Challenges

- Foundations for a PaaS-like experience?
  - Helm and Helmfile for deployment
- Developer productivity
  - Local-to-remote dev and test

HELM

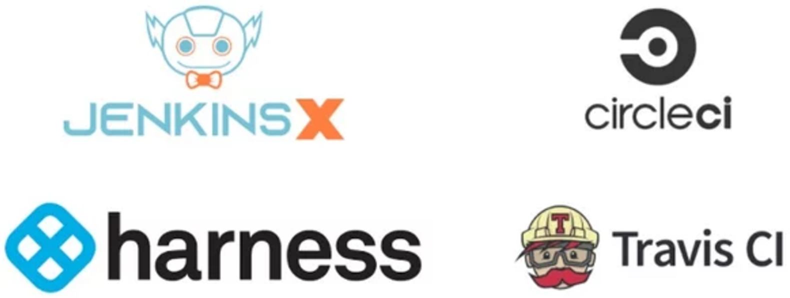
garden

KSYNC

TILT

TELEPRESENCE

Progressive Delivery: Delivery Pipelines



Progressive Delivery

Supporting the creation of pipelines that enable the automated build, verification, deployment, release, and observability

- Developers
  - Self-service interactions: automated and observable
- Platform team
  - Centralize verification of quality and security properties

https://redmonk.com/jgovernor/2018/08/06/towards-progressive-delivery/

Progressive Delivery Decisions

- Deliver any and all application changes into production as **rapidly** and as **safely** as the organisation requires
  - Pipeline practices
  - Pipeline technology

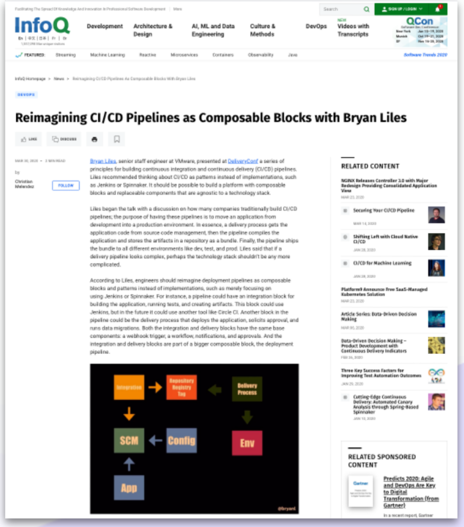
argo

JENKINS X

circleci

harness

Travis CI



Facilitating The Spread Of Knowledge And Innovation In Professional Software Development | More

Search | SIGN UP / LOGIN

InfoQ

En | 中文 | 日本 | Fr | Br

Development | Architecture & Design | AI, ML and Data Engineering | Culture & Methods | DevOps | NEW Videos with Transcripts

QCon Software Dev Conference
New York Jun 15-19, 2020
Munich Oct 19-21, 2020
SF Nov 16-20, 2020

FEATURED: Streaming | Machine Learning | Reactive | Microservices | Containers | Observability | Java | Software Trends 2020

InfoQ Homepage > News > Reimagining CI/CD Pipelines As Composable Blocks With Bryan Liles

DEVOPS

Reimagining CI/CD Pipelines as Composable Blocks with Bryan Liles

LIKE | DISCUSS

MAR 30, 2020 • 2 MIN READ

by

Christian Melendez

FOLLOW

Bryan Liles, senior staff engineer at VMware, presented at DeliveryConf a series of principles for building continuous integration and continuous delivery (CI/CD) pipelines. Liles recommended thinking about CI/CD as patterns instead of implementations, such as Jenkins or Spinnaker. It should be possible to build a platform with composable blocks and replaceable components that are agnostic to a technology stack.

Liles began the talk with a discussion on how many companies traditionally build CI/CD pipelines; the purpose of having these pipelines is to move an application from development into a production environment. In essence, a delivery process gets the application code from source code management, then the pipeline compiles the application and stores the artifacts in a repository as a bundle. Finally, the pipeline ships the bundle to all different environments like dev, test, and prod. Liles said that if a delivery pipeline looks complex, perhaps the technology stack shouldn't be any more complicated.

According to Liles, engineers should reimagine deployment pipelines as composable blocks and patterns instead of implementations, such as merely focusing on using Jenkins or Spinnaker. For instance, a pipeline could have an integration block for building the application, running tests, and creating artifacts. This block could use Jenkins, but in the future it could use another tool like Circle CI. Another block in the pipeline could be the delivery process that deploys the application, solicits approval, and runs data migrations. Both the integration and delivery blocks have the same base components: a webhook trigger, a workflow, notifications, and approvals. And the integration and delivery blocks are part of a bigger composable block, the deployment pipeline.

RELATED CONTENT

NGINX Releases Controller 3.0 with Major Redesign Providing Consolidated Application View
MAR 23, 2020

Securing Your CI/CD Pipeline
MAR 14, 2020

Shifting Left with Cloud Native CI/CD
JAN 28, 2020

CI/CD for Machine Learning
JAN 28, 2020

Platform9 Announce Free SaaS-Managed Kubernetes Solution
MAR 23, 2020

Article Series: Data-Driven Decision Making
MAR 05, 2020

Data-Driven Decision Making – Product Development with Continuous Delivery Indicators
FEB 26, 2020

Three Key Success Factors for Improving Test Automation Outcomes
JAN 29, 2020

Cutting-Edge Continuous Delivery: Automated Canary Analysis through Spring-Based Spinnaker
JAN 10, 2020

RELATED SPONSORED CONTENT

Predicts 2020: Agile and DevOps Are Key to Digital Transformation (from Gartner)

In a recent report, Gartner

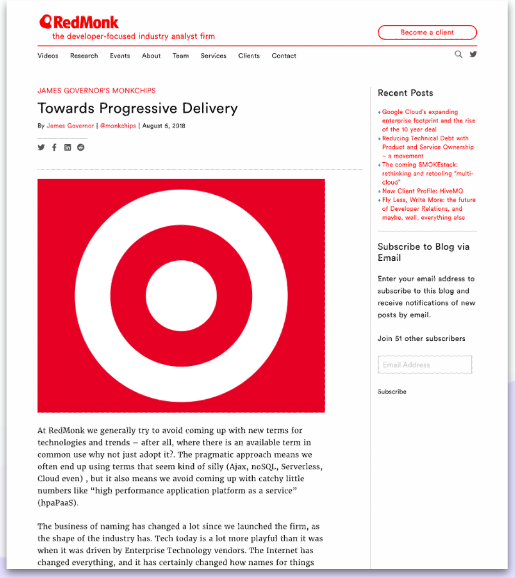
https://www.infoq.com/news/2020/03/reimagining-cicd-pipelines/

DATAWIRE

@danielbryantuk

Progressive Delivery Challenges

- Collaboration between dev, QA, and ops
- Balance one-size-fits-all vs chaos
- Make it easy to do the right thing



RedMonk
the developer-focused industry analyst firm

JAMES GOVERNOR'S MONKCHIPS

Towards Progressive Delivery

By James Governor | @monkchips | August 6, 2018

At RedMonk we generally try to avoid coming up with new terms for technologies and trends – after all, where there is an available term in common use why not just adopt it?. The pragmatic approach means we often end up using terms that seem kind of silly (Ajax, noSQL, Serverless, Cloud even) , but it also means we avoid coming up with catchy little numbers like "high performance application platform as a service" (hpaPaaS).

The business of naming has changed a lot since we launched the firm, as the shape of the industry has. Tech today is a lot more playful than it was when it was driven by Enterprise Technology vendors. The Internet has changed everything, and it has certainly changed how names for things

Recent Posts

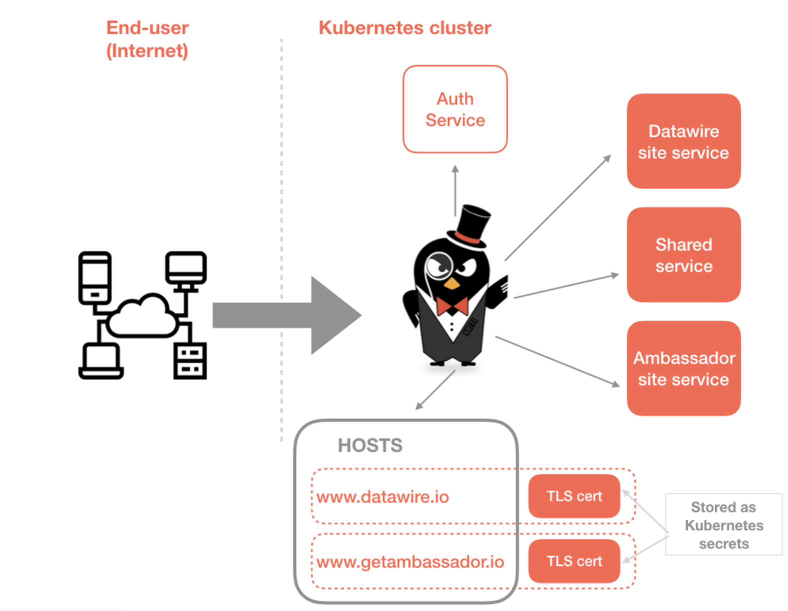
- Google Cloud's expanding enterprise footprint and the rise of the 10 year deal
- Reducing Technical Debt with Product and Service Ownership – a movement
- The coming SMOKEstack: rethinking and retooling "multi-cloud"
- New Client Profile: HiveMQ
- Fly Less, Write More: the future of Developer Relations, and maybe, well, everything else

Subscribe to Blog via Email

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Join 51 other subscribers

Edge Management: Ingress and API Gateways



DATAWIRE

Edge Management

Enable the self-service release of new functionality by developers, while maintaining stability

- Developers
  - Decentralized traffic management
  - Support NFRs e.g. authn/z, retries, and circuit breaking

- Platform
  - Centralized configuration of sane defaults
  - TLS, authn/z, and rate limiting for DDoS protection

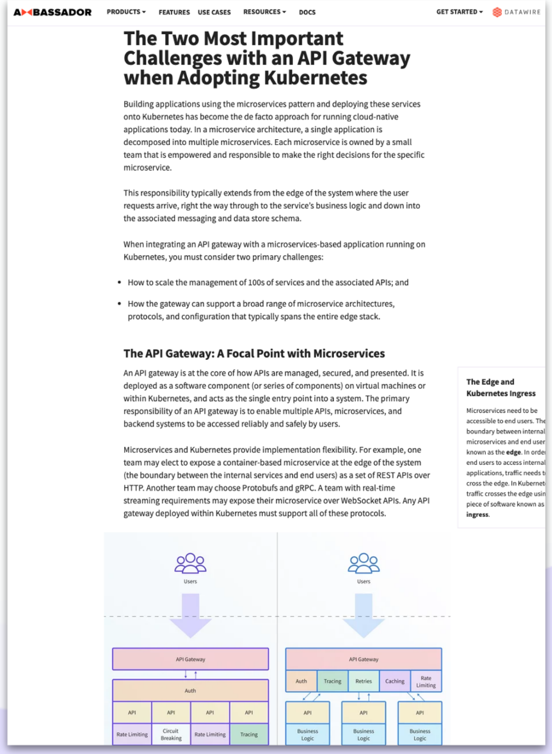
Edge Stack Decisions

- Edge technologies
  - Envoy becoming the de facto standard(?)
  - xDS APIs / Ingress v2

- Deploy/release workflows
  - Declarative (CRDs)
  - Self-service

Edge Stack Challenges

- Scaling edge management
- Supporting multiple protocols and NFRs



The Two Most Important Challenges with an API Gateway when Adopting Kubernetes

Building applications using the microservices pattern and deploying these services onto Kubernetes has become the de facto approach for running cloud-native applications today. In a microservice architecture, a single application is decomposed into multiple microservices. Each microservice is owned by a small team that is empowered and responsible to make the right decisions for the specific microservice.

This responsibility typically extends from the edge of the system where the user requests arrive, right the way through to the service's business logic and down into the associated messaging and data store schema.

When integrating an API gateway with a microservices-based application running on Kubernetes, you must consider two primary challenges:

- How to scale the management of 100s of services and the associated APIs; and
- How the gateway can support a broad range of microservice architectures, protocols, and configuration that typically spans the entire edge stack.

The API Gateway: A Focal Point with Microservices

An API gateway is at the core of how APIs are managed, secured, and presented. It is deployed as a software component (or series of components) on virtual machines or within Kubernetes, and acts as the single entry point into a system. The primary responsibility of an API gateway is to enable multiple APIs, microservices, and backend systems to be accessed reliably and safely by users.

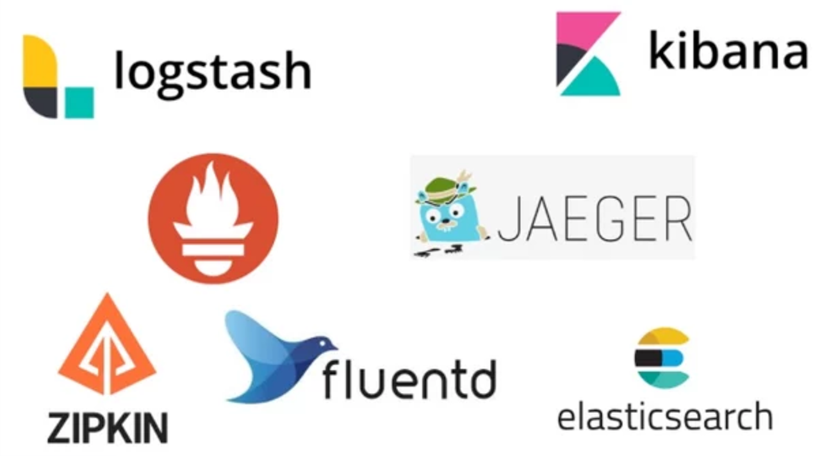
Microservices and Kubernetes provide implementation flexibility. For example, one team may elect to expose a container-based microservice at the edge of the system (the boundary between the internal services and end users) as a set of REST APIs over HTTP. Another team may choose Protobufs and gRPC. A team with real-time streaming requirements may expose their microservice over WebSocket APIs. Any API gateway deployed within Kubernetes must support all of these protocols.

The Edge and Kubernetes Ingress

Microservices need to be accessible to end users. The boundary between internal microservices and end user known as the **edge**. In orde end users to access internal applications, traffic needs t cross the edge. In Kubernet traffic crosses the edge usin piece of software known as **ingress**.

https://www.getambassador.io/resources/challenges-api-gateway-kubernetes/

Observability:
Metrics, Logging, Tracing



Observability

Support the collection and analysis of end user and application feedback directly by developers and the platform team.

- Developers
  - Enable product teams to observe and iterate against business goals and KPIs
- Platform
  - Observe and managing infrastructure, and ensure their service level objectives (SLOs) are met

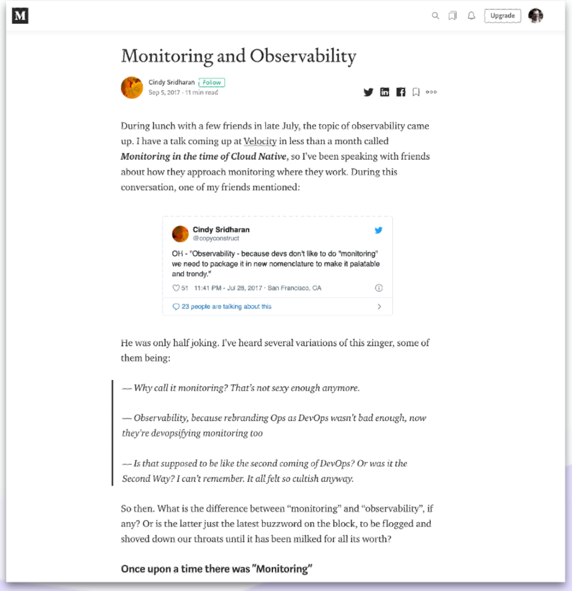
Observability Decisions

- Adoption (monitor all-the-things?)
- Technology selection (standards)
  - Metrics
  - Logging
  - Distributed tracing
- Joining the dots

honeycomb.io
logstash
kibana
JAEGER
ZIPKIN
fluentd
elasticsearch
LIGHTSTEP

Observability Challenges

- Self-service config and dashboards
- Increasing signal-to-noise
- Fault location



Monitoring and Observability

Cindy Sridharan
Sep 5, 2017 · 11 min read

During lunch with a few friends in late July, the topic of observability came up. I have a talk coming up at Velocity in less than a month called ***Monitoring in the time of Cloud Native***, so I've been speaking with friends about how they approach monitoring where they work. During this conversation, one of my friends mentioned:

> Cindy Sridharan
> @copyconstruct
>
> OH - "Observability - because devs don't like to do "monitoring" we need to package it in new nomenclature to make it palatable and trendy."
>
> ♡ 51 11:41 PM - Jul 28, 2017 · San Francisco, CA
>
> 23 people are talking about this

He was only half joking. I've heard several variations of this zinger, some of them being:

> *— Why call it monitoring? That's not sexy enough anymore.*
>
> *— Observability, because rebranding Ops as DevOps wasn't bad enough, now they're devopsifying monitoring too*
>
> *— Is that supposed to be like the second coming of DevOps? Or was it the Second Way? I can't remember. It all felt so cultish anyway.*

So then. What is the difference between "monitoring" and "observability", if any? Or is the latter just the latest buzzword on the block, to be flogged and shoved down our throats until it has been milked for all its worth?

Once upon a time there was "Monitoring"

https://medium.com/@copyconstruct/monitoring-and-observability-8417d1952e1c

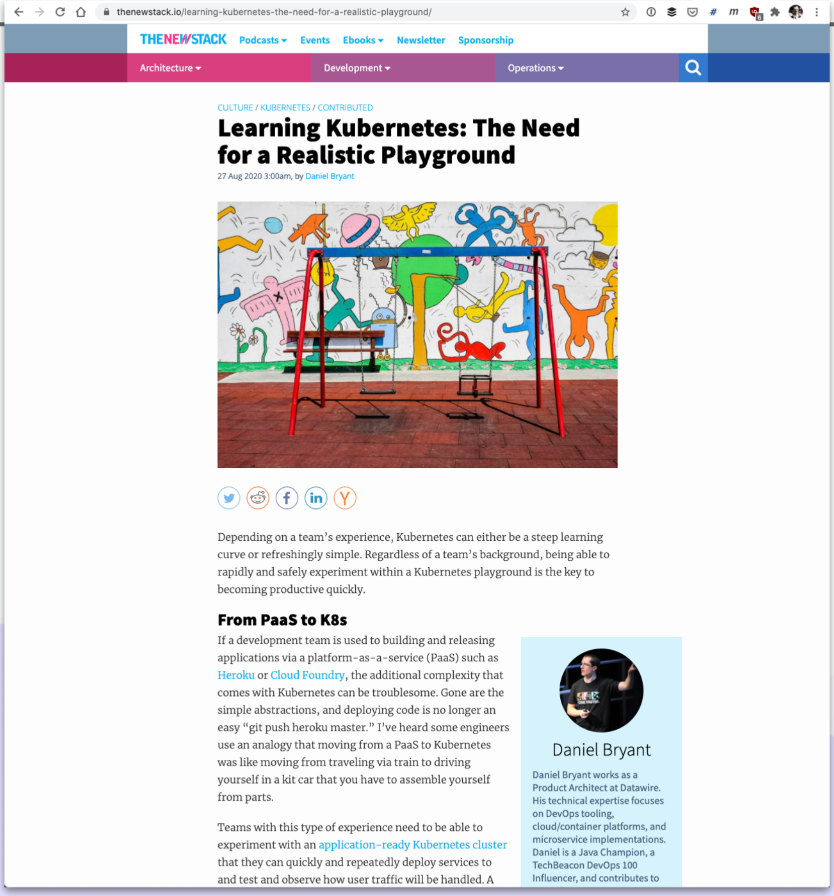
Wrapping Up

In Summary

- Being fully cloud native requires new tech and new workflows
  - Lots to be learned from full cycle development

- Creating a supporting cloud platform is essential
  - Container orchestration
  - Progressive delivery
  - Edge management
  - Observability

- Consciously design your platform & watch for antipatterns thenewstack.io/learning-kubernetes-the-need-for-a-realistic-playground/



CULTURE / KUBERNETES / CONTRIBUTED

Learning Kubernetes: The Need for a Realistic Playground

27 Aug 2020 3:00am, by Daniel Bryant

Depending on a team's experience, Kubernetes can either be a steep learning curve or refreshingly simple. Regardless of a team's background, being able to rapidly and safely experiment within a Kubernetes playground is the key to becoming productive quickly.

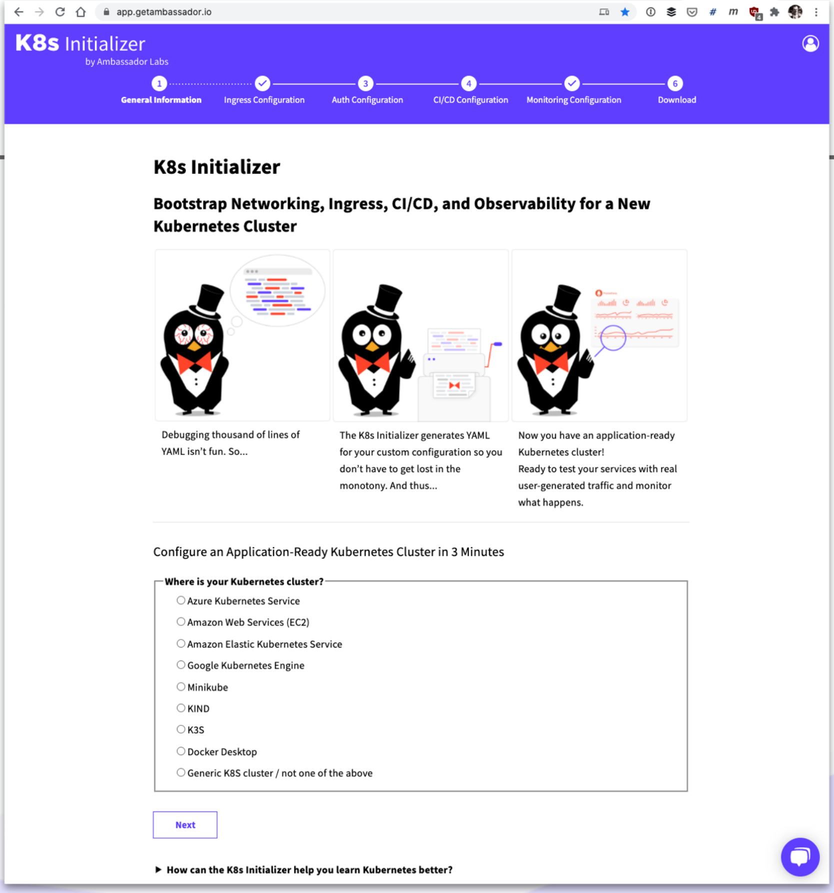
From PaaS to K8s

If a development team is used to building and releasing applications via a platform-as-a-service (PaaS) such as Heroku or Cloud Foundry, the additional complexity that comes with Kubernetes can be troublesome. Gone are the simple abstractions, and deploying code is no longer an easy "git push heroku master." I've heard some engineers use an analogy that moving from a PaaS to Kubernetes was like moving from traveling via train to driving yourself in a kit car that you have to assemble yourself from parts.

Teams with this type of experience need to be able to experiment with an application-ready Kubernetes cluster that they can quickly and repeatedly deploy services to and test and observe how user traffic will be handled. A

Daniel Bryant

Daniel Bryant works as a Product Architect at Datawire. His technical expertise focuses on DevOps tooling, cloud/container platforms, and microservice implementations. Daniel is a Java Champion, a TechBeacon DevOps 100 Influencer, and contributes to



K8s Initializer

Bootstrap Networking, Ingress, CI/CD, and Observability for a New Kubernetes Cluster

Debugging thousand of lines of YAML isn't fun. So...

The K8s Initializer generates YAML for your custom configuration so you don't have to get lost in the monotony. And thus...

Now you have an application-ready Kubernetes cluster! Ready to test your services with real user-generated traffic and monitor what happens.

Configure an Application-Ready Kubernetes Cluster in 3 Minutes

Where is your Kubernetes cluster?

○ Azure Kubernetes Service
○ Amazon Web Services (EC2)
○ Amazon Elastic Kubernetes Service
○ Google Kubernetes Engine
○ Minikube
○ KIND
○ K3S
○ Docker Desktop
○ Generic K8S cluster / not one of the above

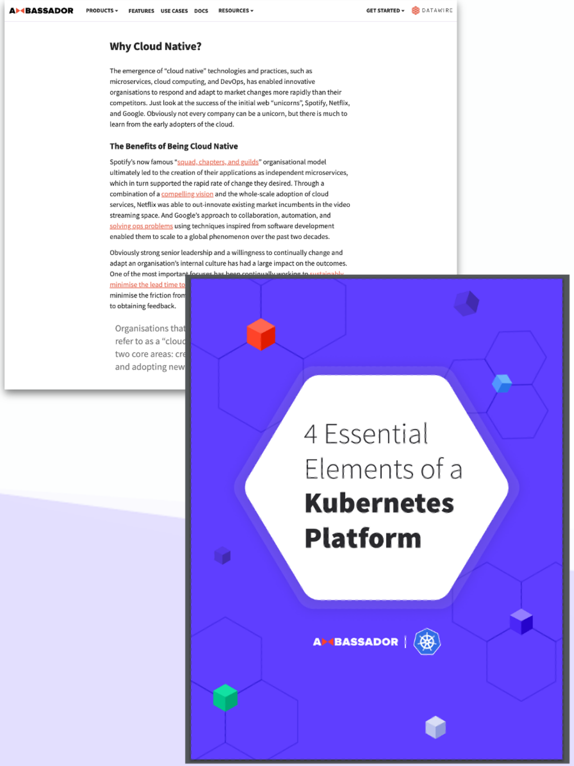
Next

▸ How can the K8s Initializer help you learn Kubernetes better?

app.getambassador.io/

DATAWIRE
@danielbryantuk

Learning More...


A▶◀BASSADOR PRODUCTS ▾ FEATURES USE CASES DOCS RESOURCES ▾ GET STARTED ▾ DATAWIRE

Why Cloud Native?

The emergence of "cloud native" technologies and practices, such as microservices, cloud computing, and DevOps, has enabled innovative organisations to respond and adapt to market changes more rapidly than their competitors. Just look at the success of the initial web "unicorns", Spotify, Netflix, and Google. Obviously not every company can be a unicorn, but there is much to learn from the early adopters of the cloud.

The Benefits of Being Cloud Native

Spotify's now famous "squad, chapters, and guilds" organisational model ultimately led to the creation of their applications as independent microservices, which in turn supported the rapid rate of change they desired. Through a combination of a compelling vision and the whole-scale adoption of cloud services, Netflix was able to out-innovate existing market incumbents in the video streaming space. And Google's approach to collaboration, automation, and solving ops problems using techniques inspired from software development enabled them to scale to a global phenomenon over the past two decades.

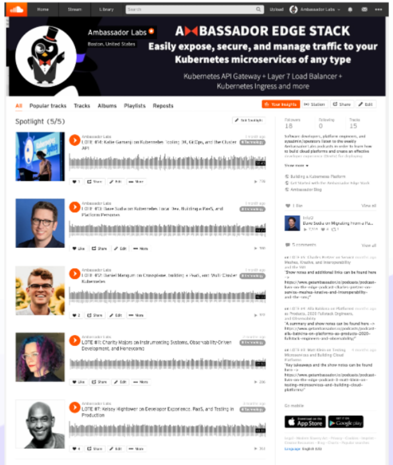
Obviously strong senior leadership and a willingness to continually change and adapt an organisation's internal culture has had a large impact on the outcomes. One of the most important focuses has been continually working to sustainably minimise the lead time to
minimise the friction from
to obtaining feedback.

Organisations that
refer to as a "cloud
two core areas: cre
and adopting new

4 Essential Elements of a **Kubernetes Platform**

A▶◀BASSADOR

Read "Building a Kubernetes Platform":
https://www.getambassador.io/learn/building-kubernetes-platform/



Subscribe to podcasts:
https://www.getambassador.io/podcasts/

Follow us on Twitter:
https://twitter.com/getambassadorio

Ambassador CNCF Incubations proposal:
https://github.com/cncf/toc/pull/435

DATAWIRE
@danielbryantuk