
AWS
re:Invent

CON302

Building resilient Kubernetes deployments with AWS App Mesh

Jesse Butler

Senior Developer Advocate

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

Modern application deployments

Deployment options with Kubernetes

Leveraging abstractions and automation

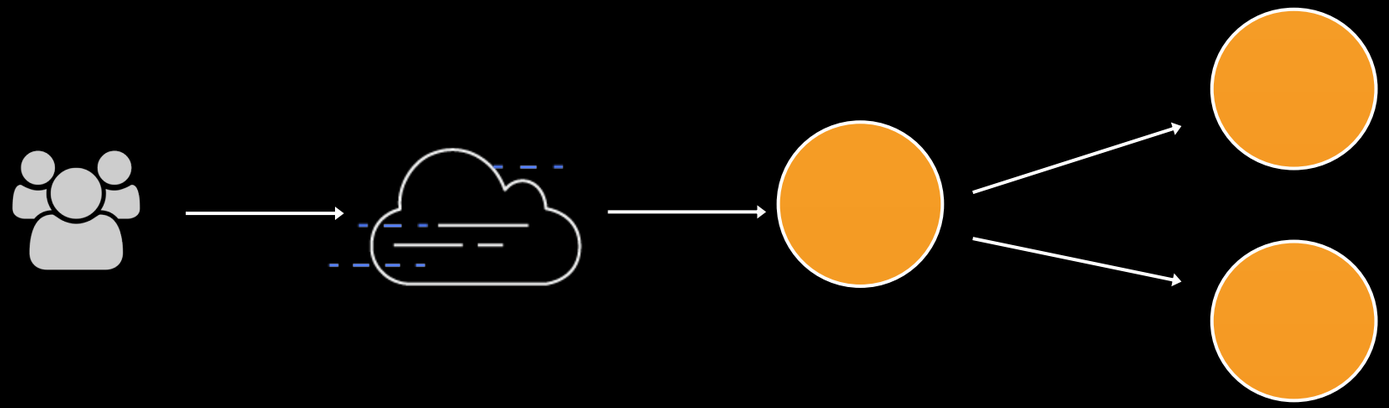


AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

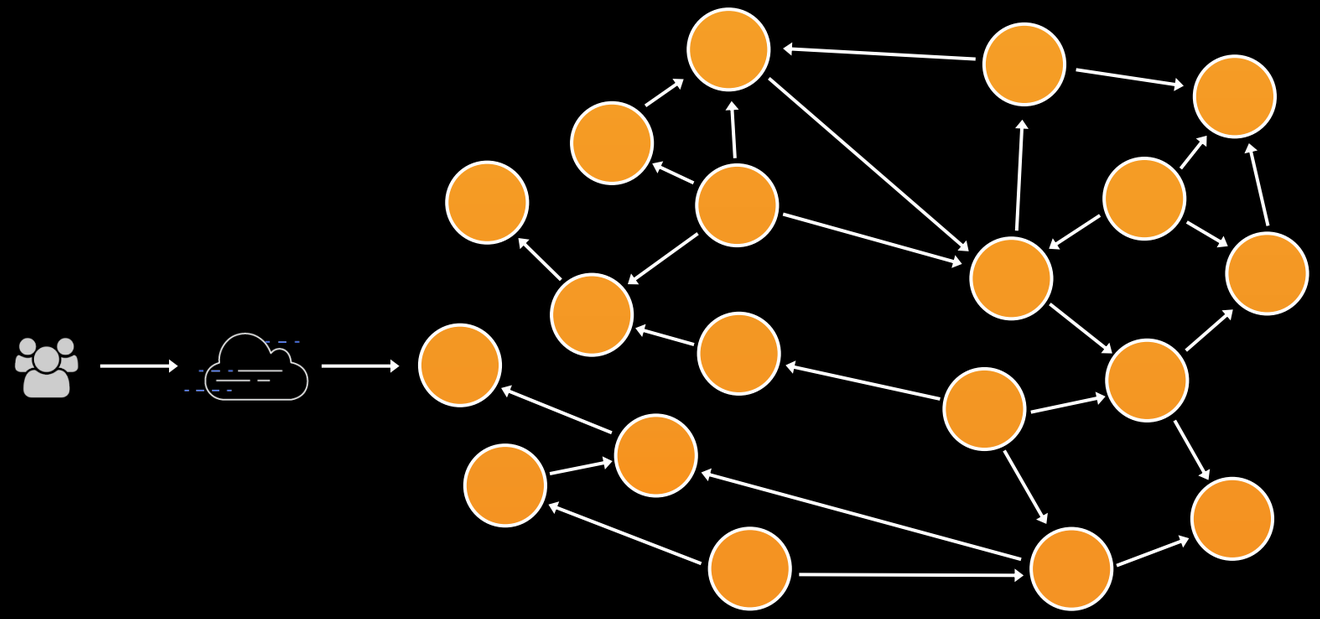
First things first

Some things work perfectly well on a smaller scale



Plan for scale

When the application scales, you may need to adapt

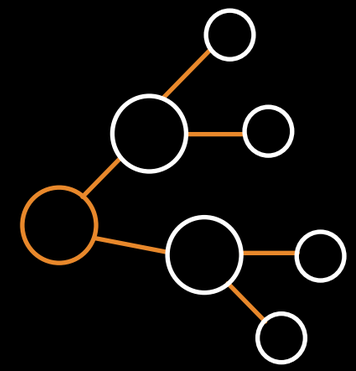


Design for scale

- Agility
- Sanity
- Resilience

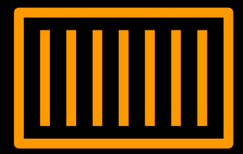
Microservices for agility

- Separation of concerns
  - Impact only the functionality you change
- More effective iteration
  - Build less code, test less code
- Maintain quality over integrations
  - Transparently modify components through well-defined APIs



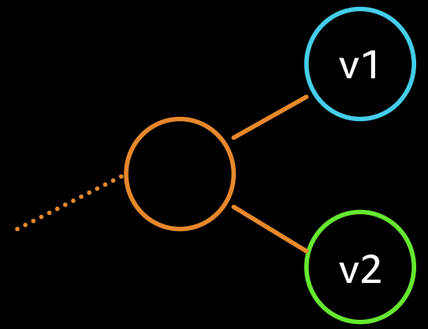
Containers for sanity

- Separation of concerns
  - Impact as few application processes as possible
- More effective iteration
  - Behavior of services is consistent through all phases
- Maintain quality over integrations
  - Immutable self-contained compute environments



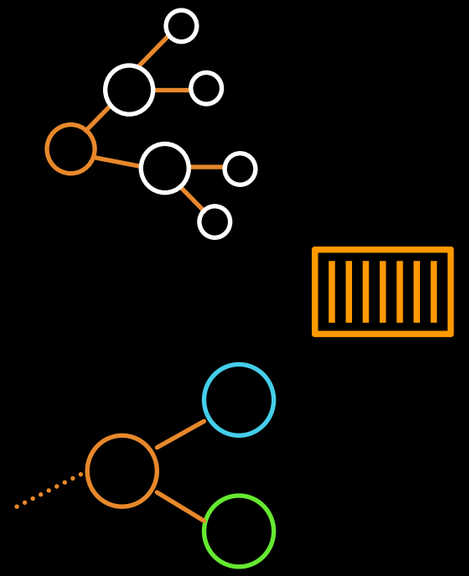
Modern deployments for resilience

- Separation of concerns
  - Impact as few system components as possible
- More effective iteration
  - Move changes through the system incrementally
- Maintain quality over integrations
  - Modular, incremental updates are transparent

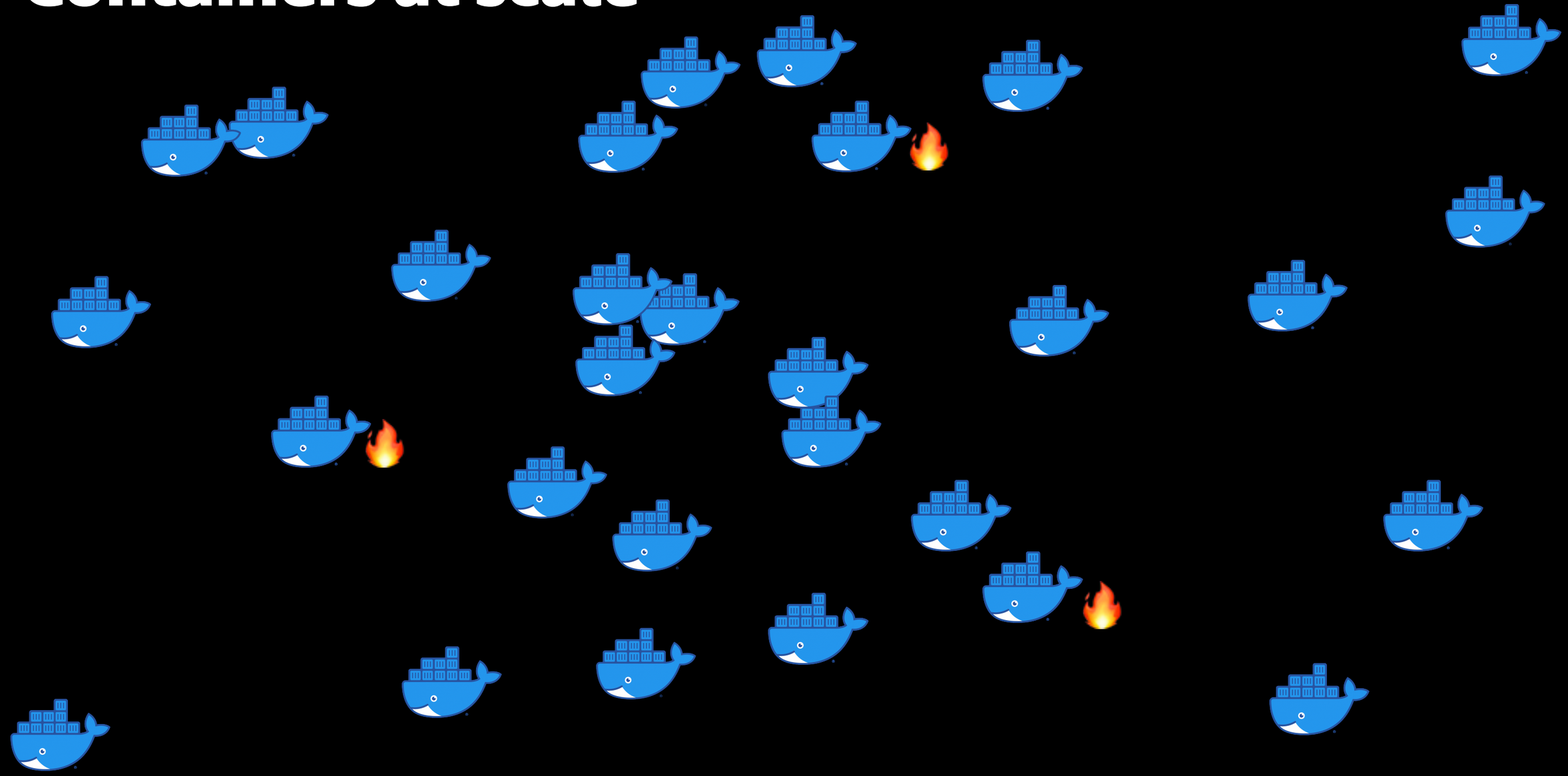


Cloud native table stakes

- Separation of concerns
- Effective iteration
- Quality over integrations



Containers at scale



Managed container orchestration with AWS

Amazon Elastic Container Service (Amazon ECS)

A fully managed container orchestration service



Amazon Elastic Kubernetes Service (Amazon EKS)

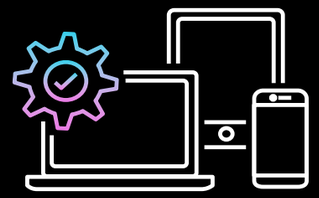
A fully managed container orchestration service



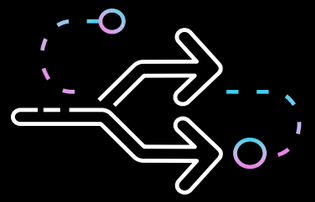
Amazon EKS

Amazon EKS is upstream Kubernetes, with the hard parts handled by AWS

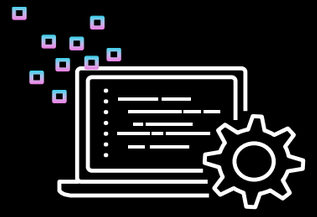
Fully managed control plane, managed data plane, AWS Fargate



Production ready



Native and upstream



OSS contribution



Seamless integrations

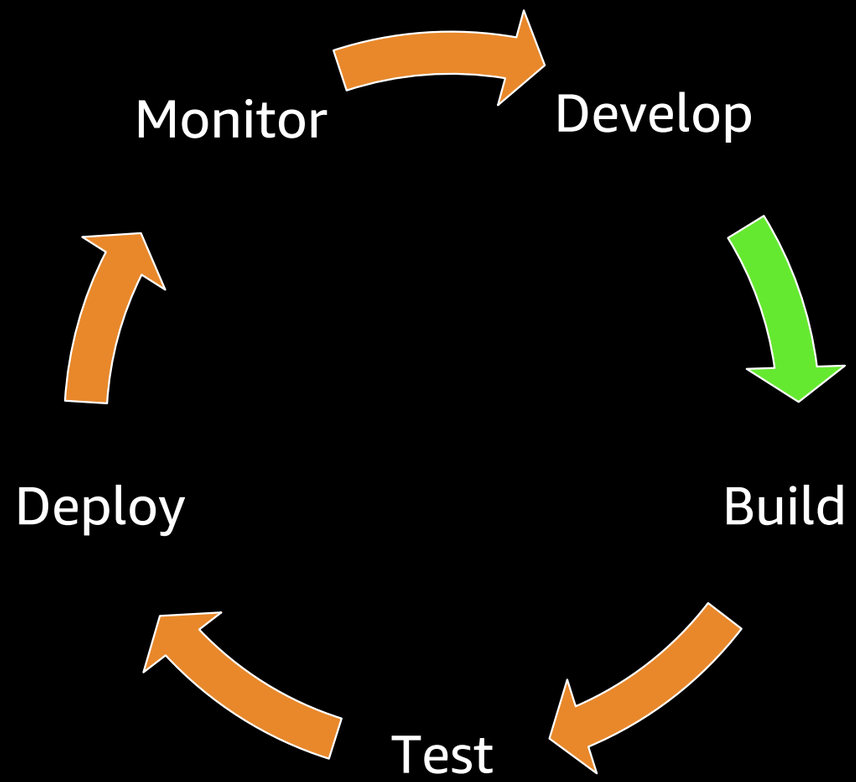
Stage is set

- ✅ Designing with microservices
- ✅ Building and shipping with containers
- ✅ Orchestration service adopted

Enter stage right: Lifecycle management

Since the dawn of time
(or at least January 1, 1970)

- Write code
- Build code
- Ship code
- Run code
- Break code
- goto Write code



Code and more code

Thinking beyond code … or it’s all “code”🤔

- Service code
- Dockerfiles
- Kubernetes manifests
- Goal: own less code

How to tell the truth

- Your code is version controlled
- You build this code into images
- You use these images in deployments
- Correlation between code and running bits is vital

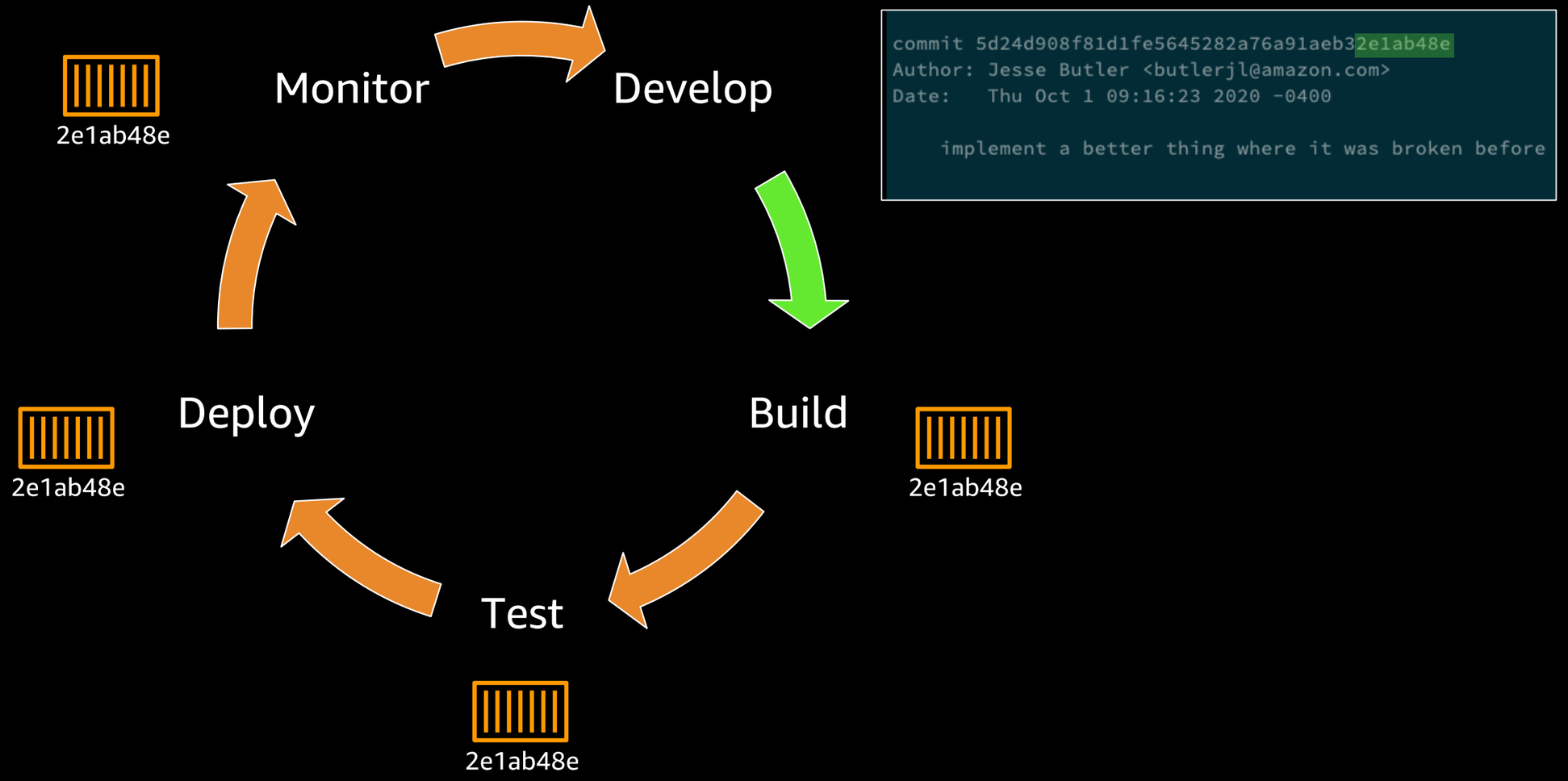
Container image as immutable artifact

Correlation is vital

Meaningful tags are meaningful

- Git commit ID
- Semantic Versioning
- Use vanity tags only if your teams can easily determine actual version

Lifecycle management with containers



Native Kubernetes rolling updates

This default strategy can work really well, depending upon your needs

```
Name:                   weather-svc
Namespace:              simpleweather
CreationTimestamp:      Mon, 01 Oct 2020 21:10:33 -0400
Labels:                 app=weather-svc
Annotations:            deployment.kubernetes.io/revision: 1
                        kubectl.kubernetes.io/last-applied-configuration:
                          {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations
":{},"labels":{"app":"weather-svc"},"name":"weather-svc","namespace":"...
Selector:               app=weather-svc
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
..
```

Kubernetes RollingUpdate strategy

- New pods are created while old ones are killed off
- Containers are killed with SIGTERM
- The scheduler will continue to throw work at the underlying pods
- Maybe before new pods are actually ready
- LoadBalancer config may lag (definitely will lag)

```
$ kubectl set image deployment/weather-svc weather-svc=weather-svc:2e1ab48e --record
# kubectl rollout status deployment.v1.apps/weather-svc
  Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
```

Kubernetes rolling updates can work well

- Implement Readiness Probes
- Ensure your service can handle a SIGTERM
  - Including any subprocesses 🧟
- Specify preStop lifecycle hooks
- Plan for upstream / downstream interruptions

```
$ kubectl set image deployment/weather-svc weather-svc=weather-svc:2e1ab48e --record
# kubectl rollout status deployment.v1.apps/weather-svc
  deployment.apps/weather-svc successfully rolled out
```

This only gets us so far

- A deployment cannot have more than one version update
- Simply migrating from one state to another
- Container offlining is not ordered, time your sidecars
- Start, pause, and rollback are the only controls

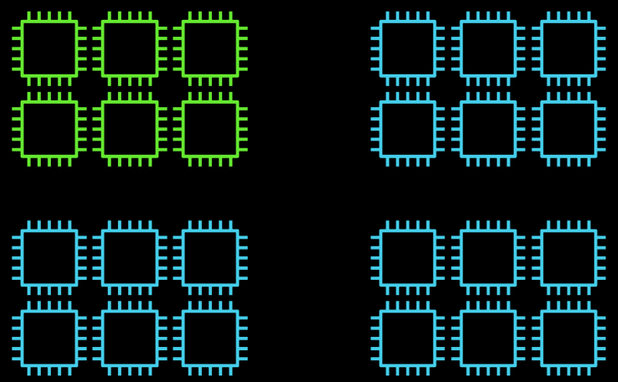
Sometimes that’s all you need, but plan for scale

Taking control of deployments

There are many ways to implement zero-downtime deployments

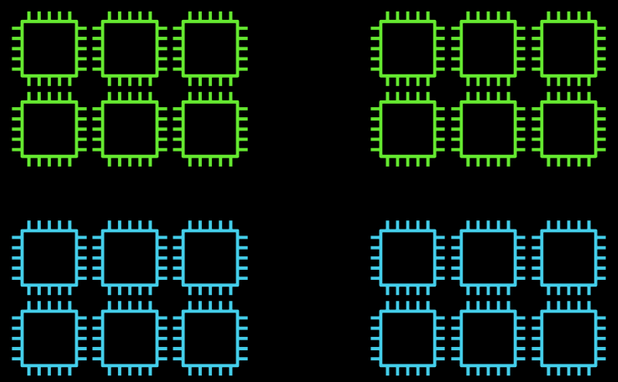
Rolling deployments

Release candidate gradually replaces the current prod over time



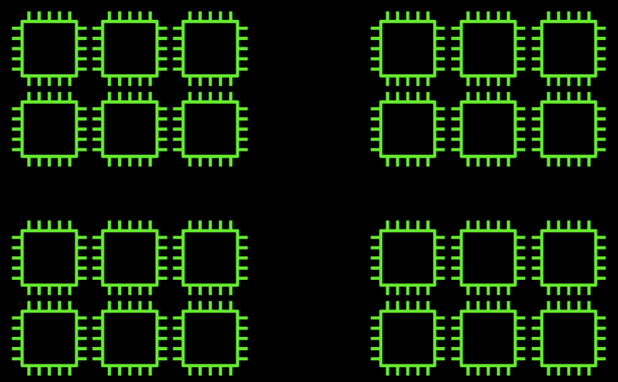
Rolling deployments

Release candidate gradually replaces the current prod over time



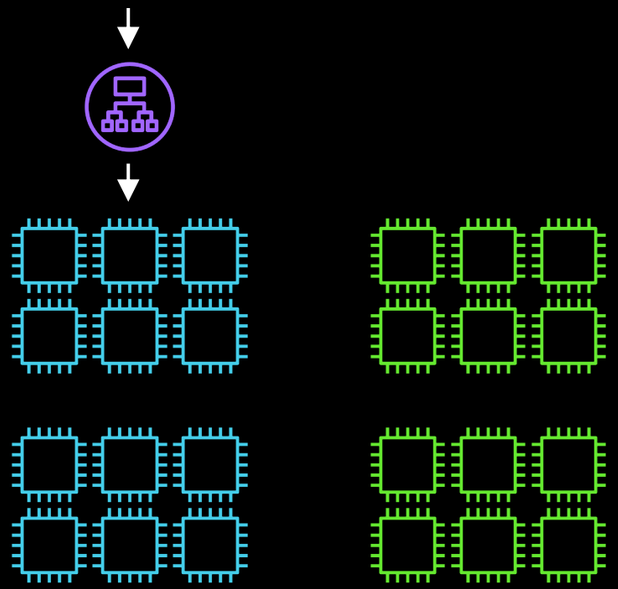
Rolling deployments

Release candidate gradually replaces the current prod over time



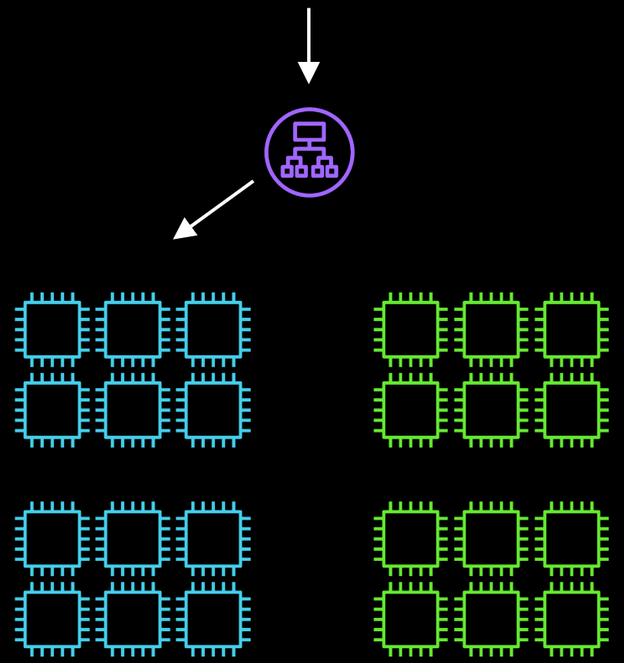
Blue / green deployments

Two identical environments exist, one is the current production



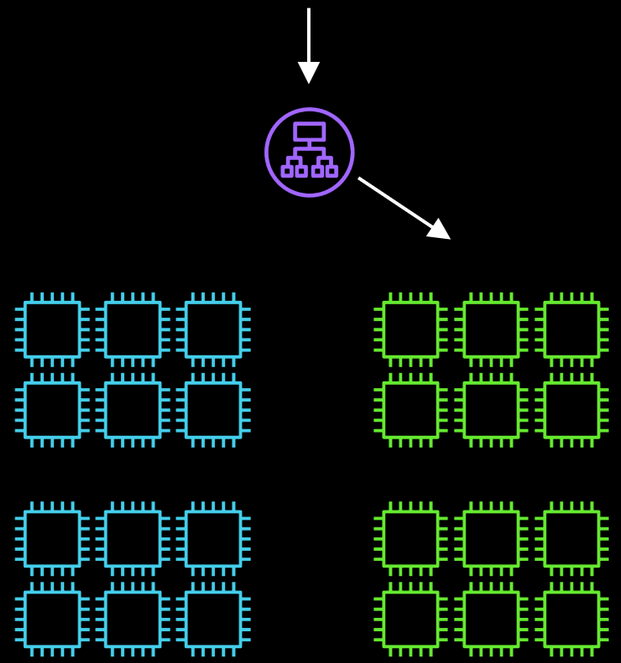
Blue / green deployments

Once tested, traffic is shifted to the new release



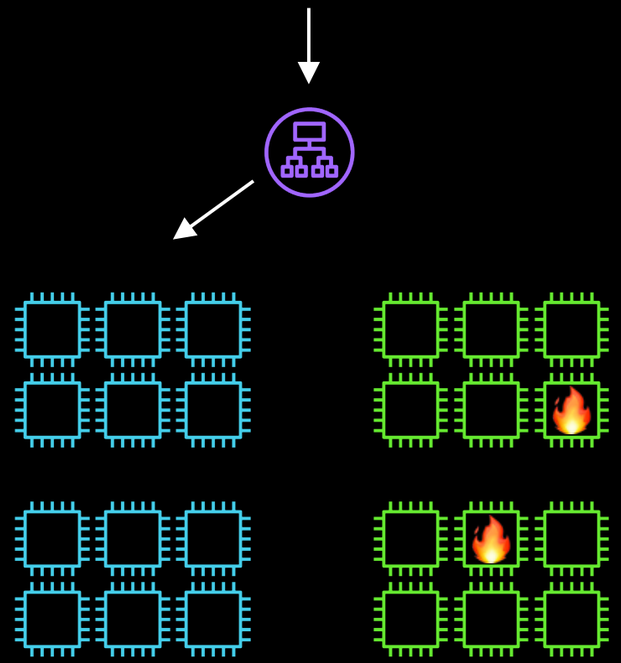
Blue / green deployments

Now the next release can be deployed and tested on blue



Blue / green deployments

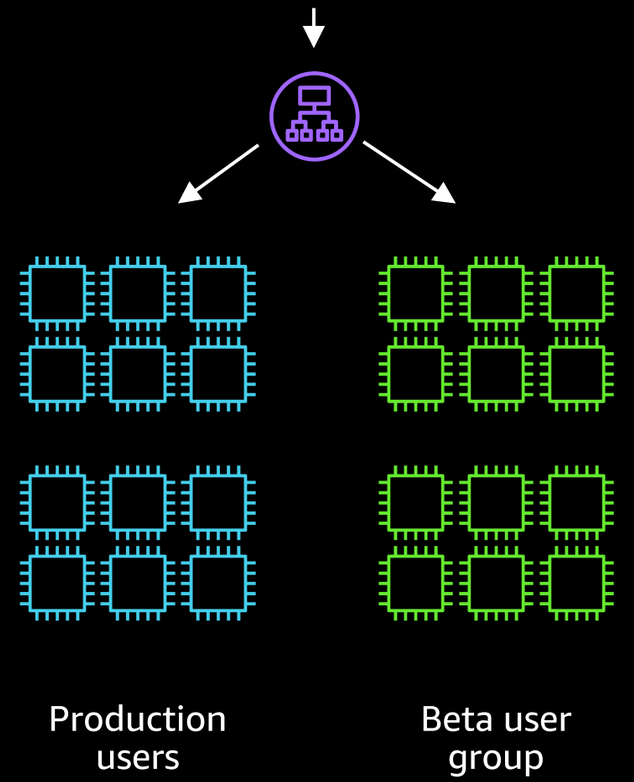
In case of failure, traffic can be switched back



A/B test configurations

Useful for UX and beta testing, you can test between two releases

For example, Beta testers may get access to new features this way

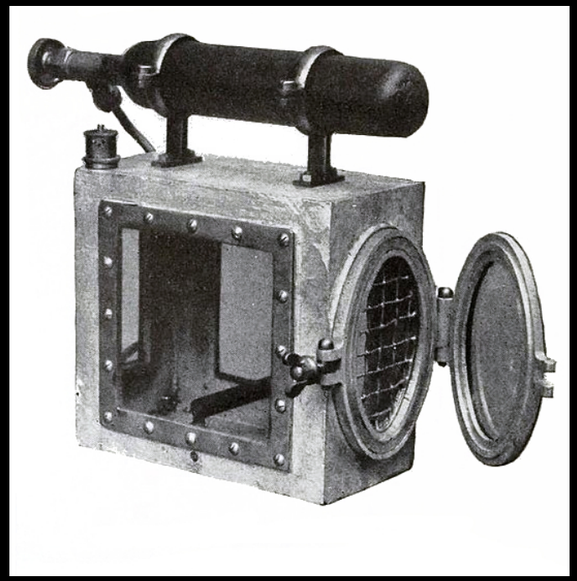


Modern container deployments

- We no longer work with racks of machines
- We have any number of ways to manage traffic
- It’s affordable to incrementally scale up resources
- It is simple to rollout and monitor a deployment in Kubernetes
- The wheel need not be re:Invent’d (ugh)

Case in point: Canary releases

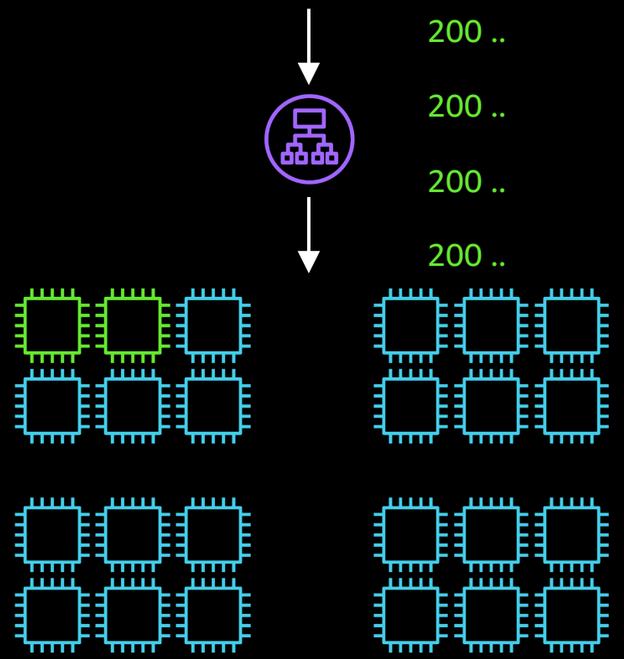
- Incremental deployment
- Start small, increase over time
- Monitor health over entire process
- Rollback if things go sideways
- Releases are transparent, without downtime



AWS does not promote nor condone the use of live canaries for testing software deployments

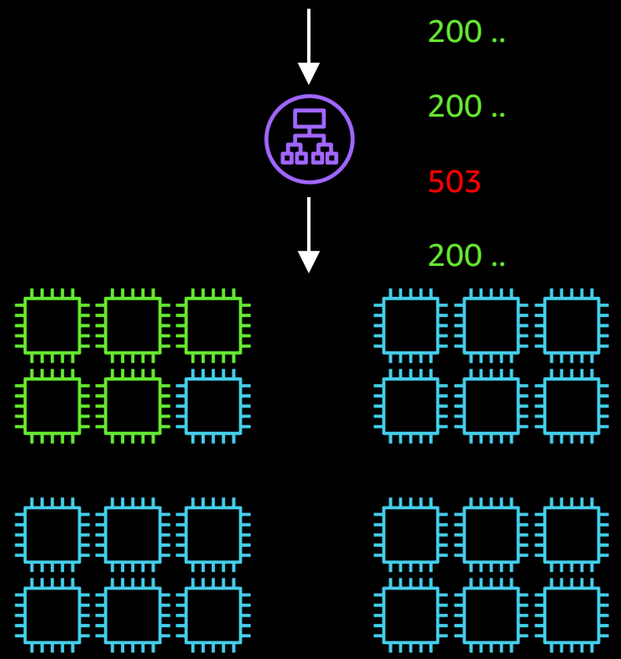
Canary releases

- Explicitly controlled incremental rollout of release candidate



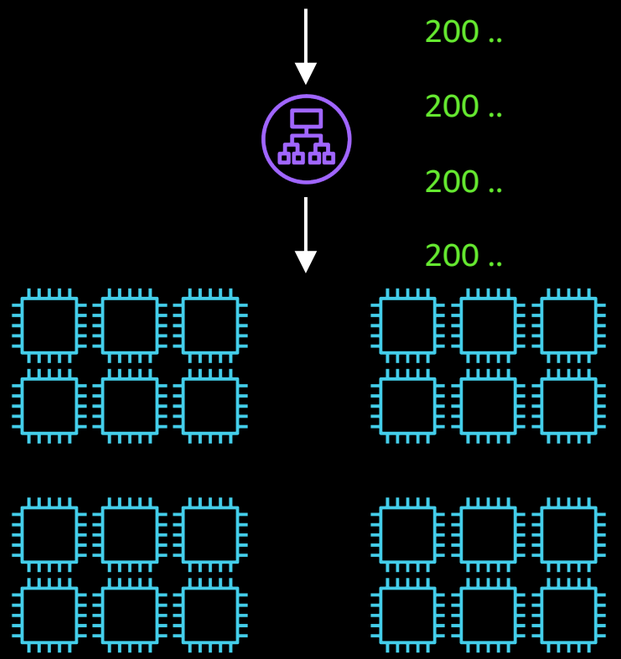
Canary releases

- Explicitly controlled incremental rollout of release candidate
- All components are actively monitored during rollout



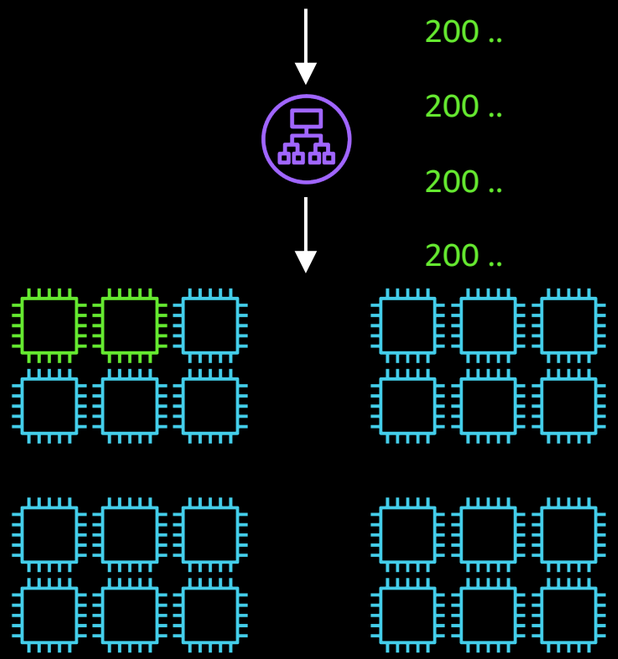
Canary releases

- Explicitly controlled incremental rollout of release candidate
- All components are actively monitored during rollout
- Rollback is triggered if health deteriorates



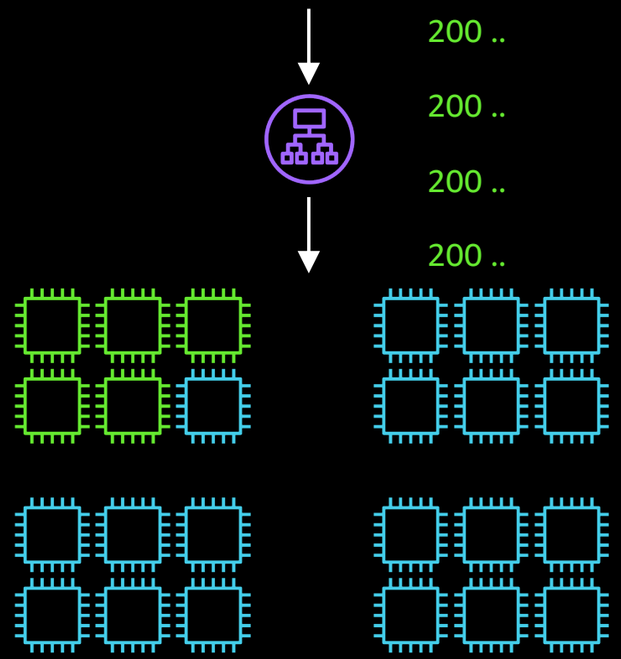
Canary releases

- Each release is a complete cycle



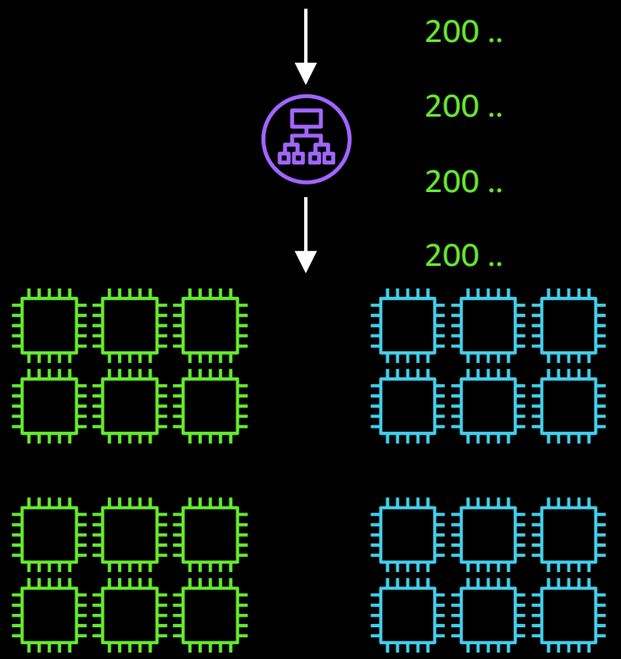
Canary releases

- Each release is a complete cycle
- Once issues are triaged and resolved, a new release is kicked off



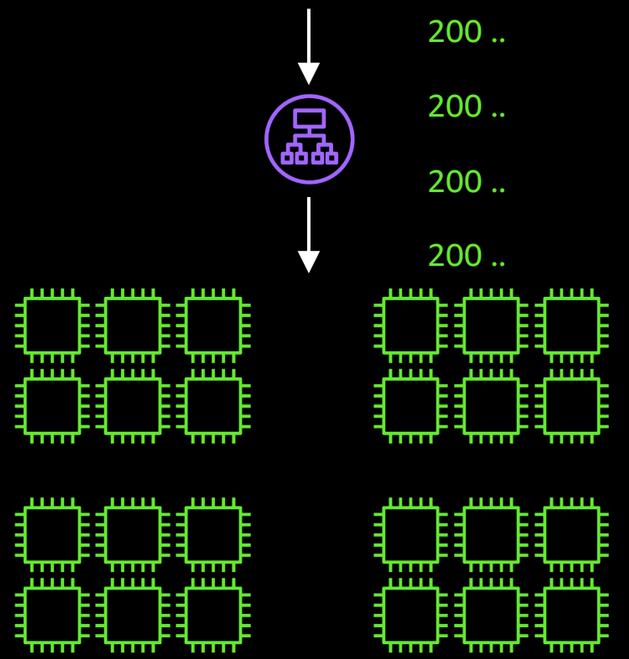
Canary releases

- Each release is a complete cycle
- Once issues are triaged and resolved, a new release is kicked off
- And as long as things are healthy…



Canary releases

- Each release is a complete cycle
- Once issues are triaged and resolved, a new release is kicked off
- And as long as things are healthy, the release completes



Where this fits in the pipeline

CI / CD is more than two things

- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- Continuous Verification
- ..

Where this fits in the pipeline

CI/CD is more than two things

- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- Continuous Verification
- ..

Building deployments

- Map deployments to commits
- Manage versioned deployments
- Configure granular traffic control
- Monitor health of system and components
- Keep the door open for a rollback if things go sideways
- Automate everything

Is this DevOps?

Own less code

You own your business logic, including its runtime configuration and deployment specification

Unless there is a very good reason to do otherwise, lean on the systems you have chosen and managed services for everything else

Logical goal: GitOps

Operational methodology for cluster and application management

- Git is the source of truth for your application
- Version-controlled immutable deployments
- Declarative configuration of Kubernetes and Infrastructure-as-code
- Extend CI / CD patterns, resulting in hands-off deployments
- Automate everything between git commit and running in prod

Logical goal: GitOps

All teams own this workflow; it must be automated

But you really should own less code

AWS App Mesh

App Mesh is a fully managed service mesh

Provides an array of modern infrastructure solutions

- Intelligent, application-aware traffic management
- Security and policy management and enforcement
- Observability support for a number of solutions



Why service mesh?

None of these solutions are unique to a service mesh

Service mesh rethinks how we support these solutions

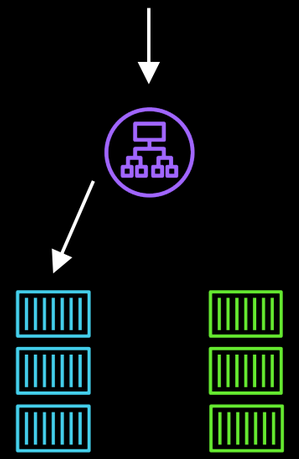
Reality check

Use Load Balancers

- They are already in place
- Weighted target groups for the win!

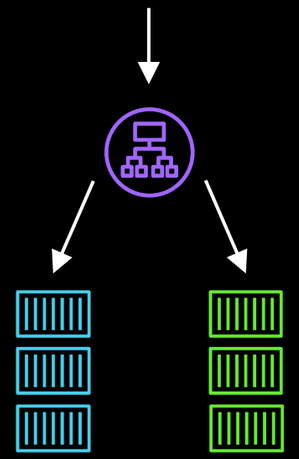
Just use load balancers

- Blue is current release, green is ready to go



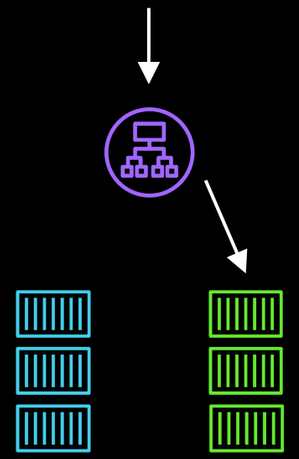
Just use load balancers

- Blue is current release, green is ready to go
- Route traffic to green, offline blue



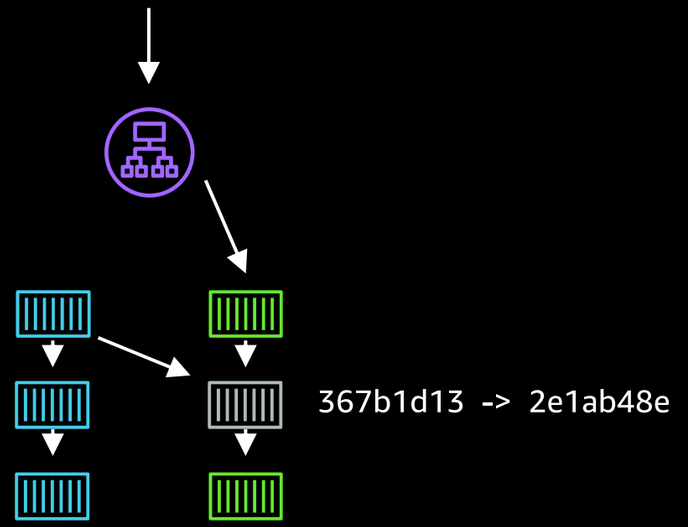
Just use load balancers

- Blue is current release, green is ready to go
- Route traffic to green, offline blue
- Rejoice in this simplicity



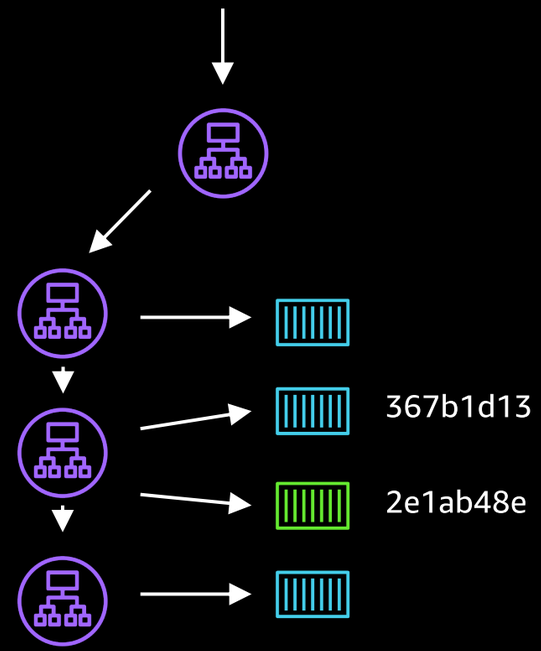
Just use load balancers

- This can work great for you
- How do we update a service in the middle?
- Either all services invoke topcalls through the load balancer, or…



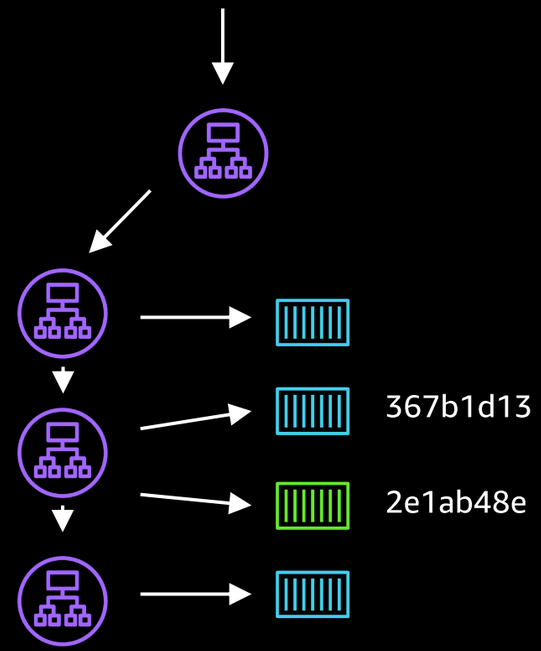
Just use load balancers

- We provision a load balancer per service



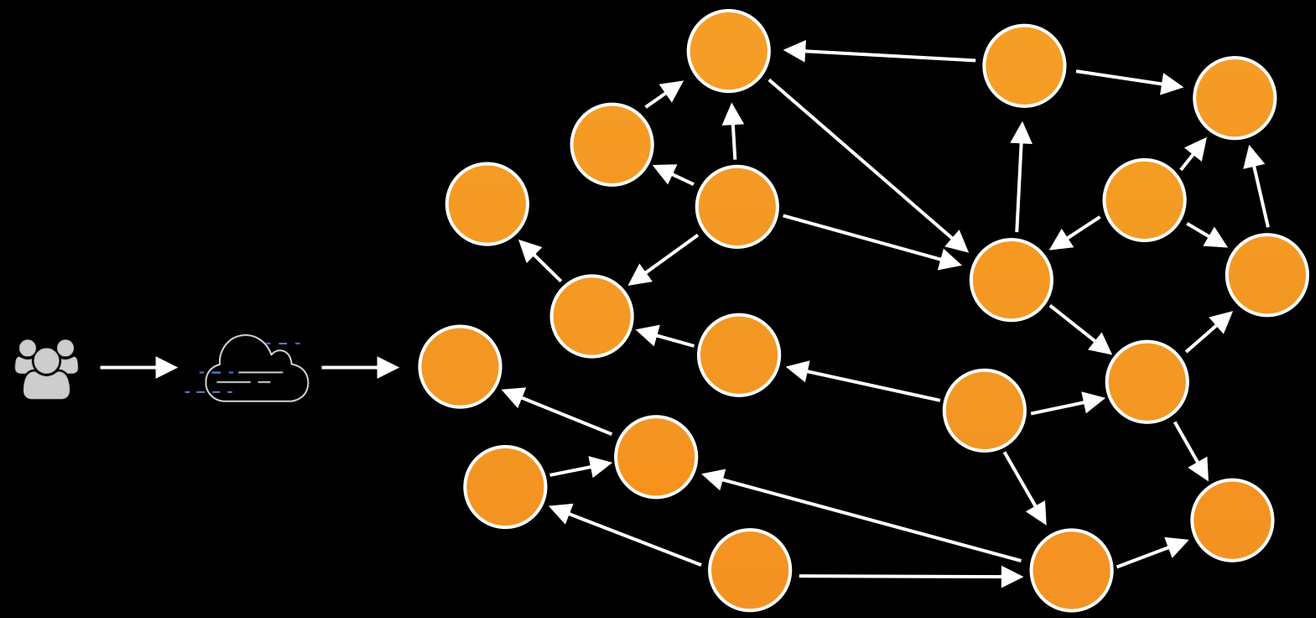
Just use load balancers

- And this still might work great for you



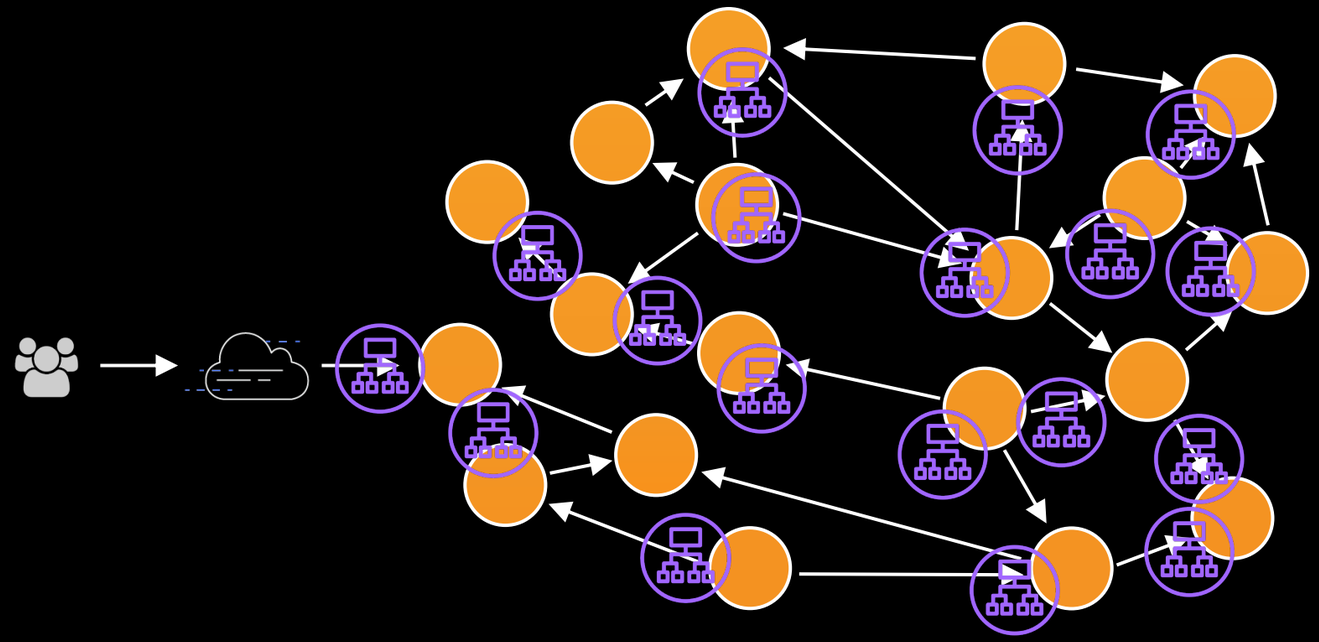
Just use load balancers

- But planning for scale…



Just use load balancers

- You don’t really want to provision a load balancer per service



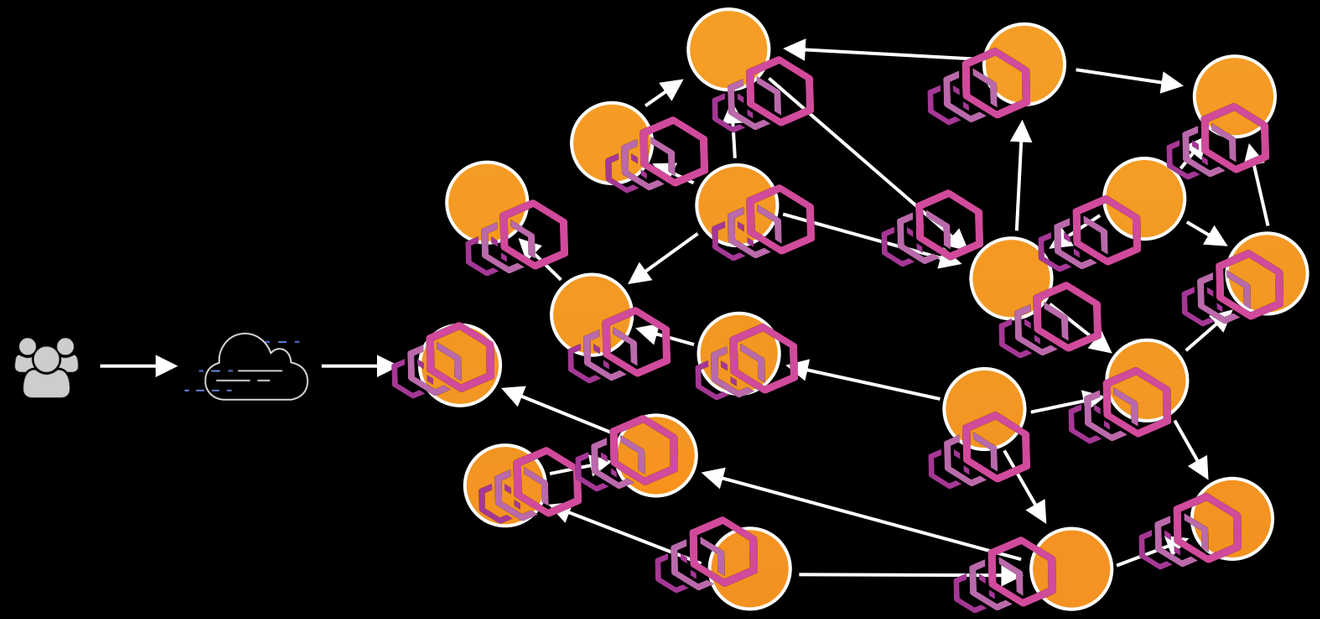
Let’s use proxy sidecars

- Highly performant cloud native proxy
- Super efficient, small footprint
- Client-side load balancing
- Retries, timeouts, rate limiting
- Observability for L7 traffic
- Support for HTTP, HTTP/2, gRPC
- Rich and robust APIs for configuration



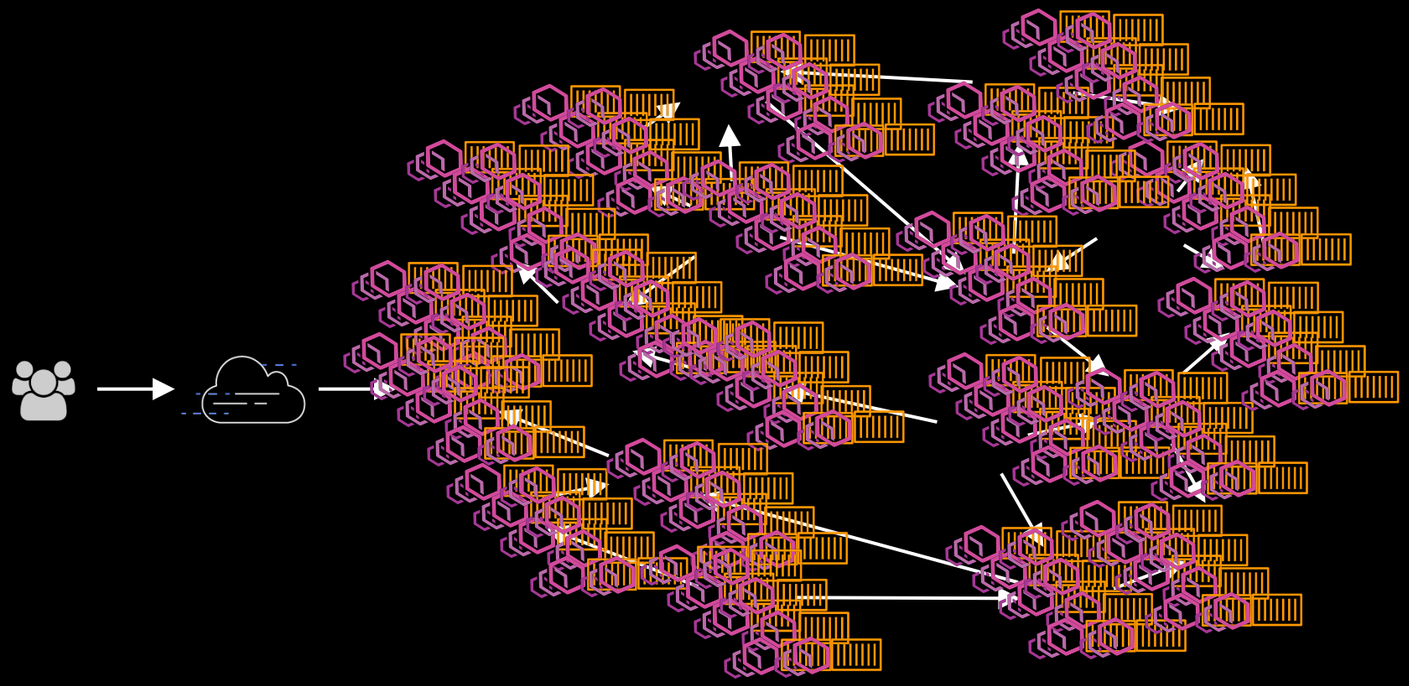
Ok, then use proxy sidecars

Well… that gets difficult quickly



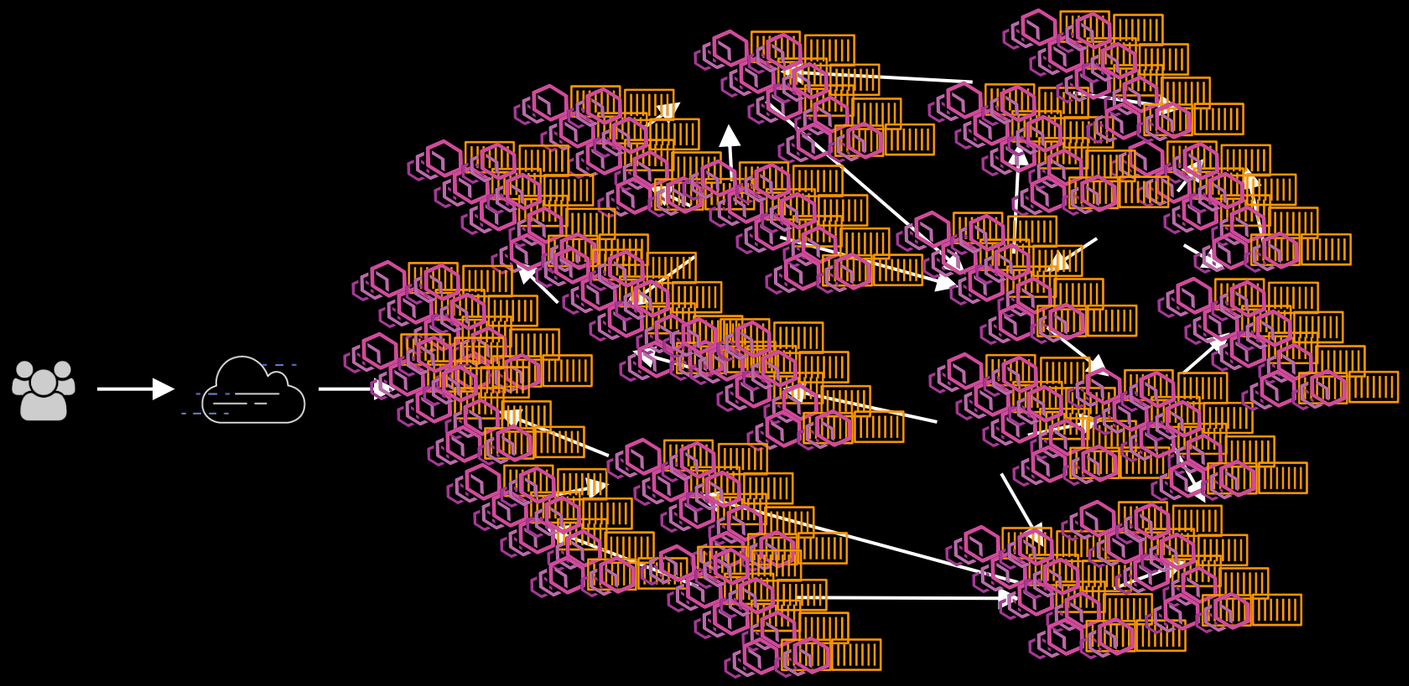
Ok, then use proxy sidecars

Don’t worry, it gets worse with containers



Ok, then use proxy sidecars

Goal: Own less code



The case for a service mesh

Service management, configuration, and connectivity

Many services, many instances of each service

Many ephemeral moving parts

Proxies are the answer, but they are complex to configure

Service mesh is an implementation detail

This is why we built App Mesh – it facilitates the management of a powerful data plane through a fully managed, easy to use control plane

App Mesh provides easy-to-use constructs and does the heavy lifting of deploying and configuring Envoy proxy instances for you

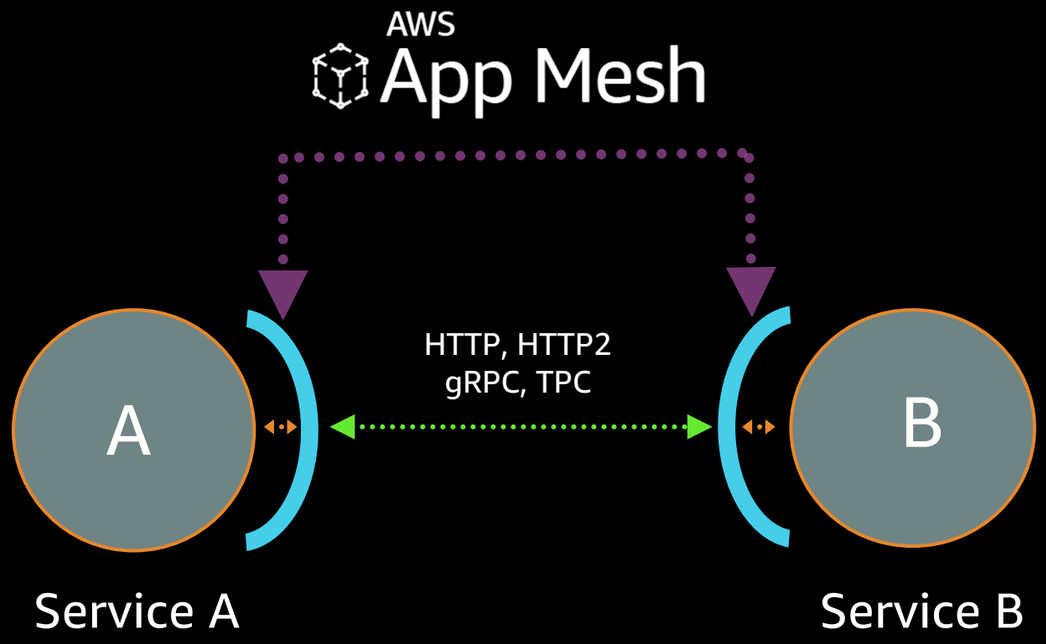
App Mesh brings a lot to the party 🎉

App Mesh is a fully managed service mesh; this bears repeating 🐻

Provides an array of modern infrastructure solutions

- Intelligent, application-aware traffic management
- Security and policy management and enforcement
- Observability support for a number of solutions

How does App Mesh work?



Control plane

Translates intent to proxy config and distributes to all proxies

Data plane

Proxies sit between all services, managing and observing traffic

App Mesh control plane constructs

Virtual service
- A configuration abstraction of a service in your cluster



Virtual router
- Defines traffic routes for one or more virtual services

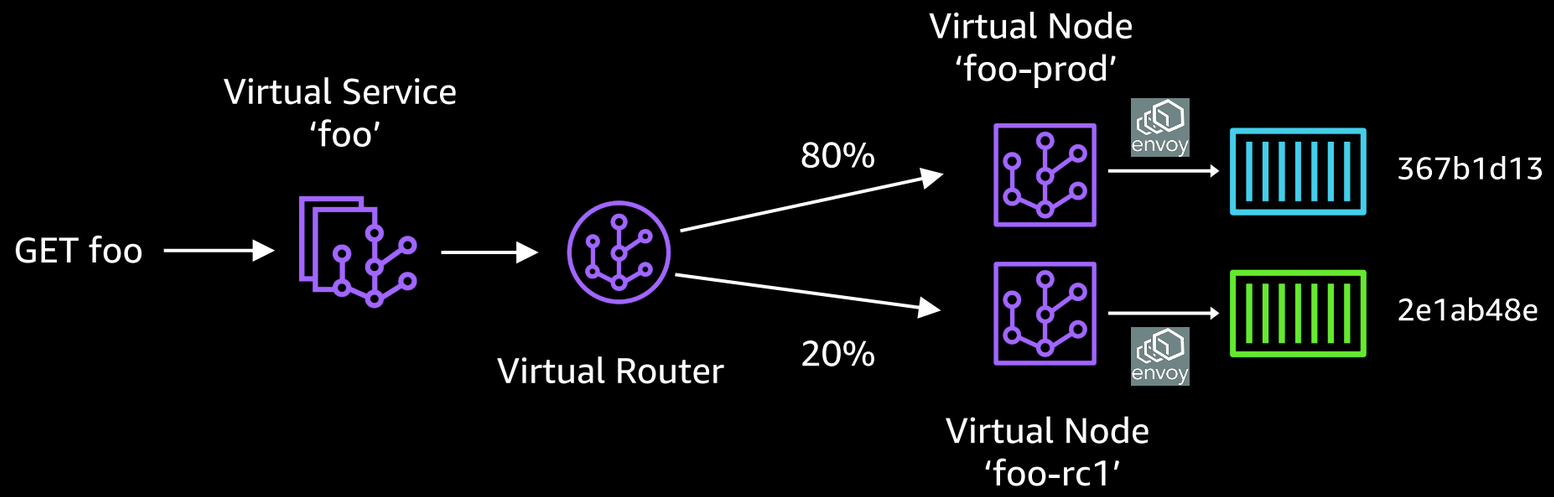


Virtual node
- A logical pointer to a discoverable service, for example, a Kubernetes service in your cluster



Weighted traffic in App Mesh

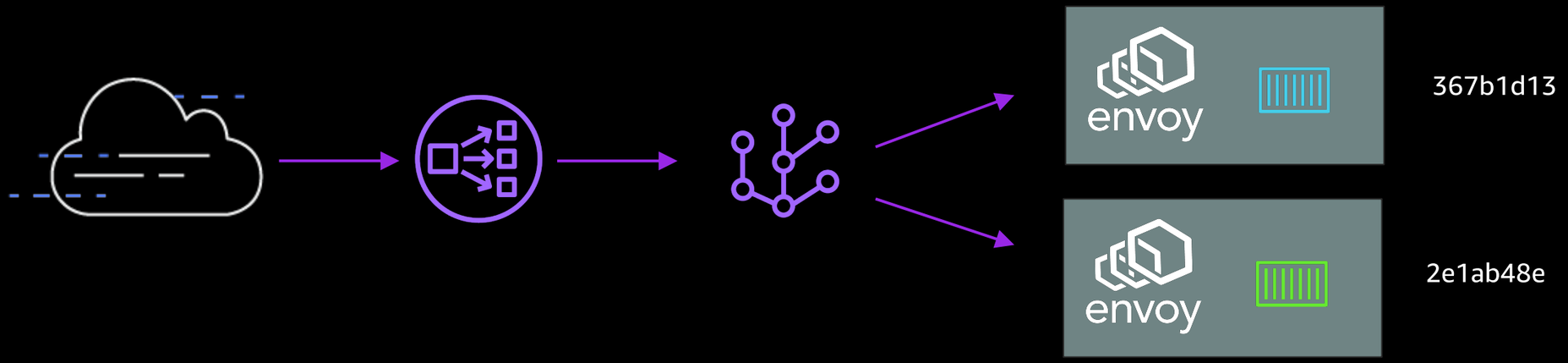
- Specify a VirtualService for ‘foo’
- Specify VirtualNodes for two versions of the backend ‘foo’
- Configure Routes with weights



App Mesh data plane

Envoy proxy containers run sidecar in your service pods

App Mesh manages their configuration via fully managed control plane



AWS App Mesh controller for Kubernetes

An open source Kubernetes controller for managing App Mesh resources from Kubernetes

- Custom resources map to all App Mesh API objects
- Automatic injection of Envoy proxy sidecars
- Controller monitors changes in these resources and reflects into App Mesh on your behalf



https://github.com/aws/aws-app-mesh-controller-for-k8s

A building block: Traffic shifting

Putting this all together

- Specify App Mesh configuration
- Apply to your cluster
- Rejoice in this simplicity

```
apiVersion: appmesh.k8s.aws/v1beta2
kind: VirtualRouter
metadata:
  namespace: foobles
  name: foobles-path-routes
spec:
  listeners:
    - portMapping:
        port: 8080
        protocol: http
  routes:
    - name: route-foo
      httpRoute:
        match:
          prefix: /foo
        action:
          weightedTargets:
            - virtualNodeRef:
                name: foo-prod
              weight: 80
            - virtualNodeRef:
                name: foo-rc1
              weight: 20
---
```

Downsides?

Let’s be fair

- More to learn
- More moving parts
- Latency, cost, and complexity

You need to solve for these problems regardless

- Minimize what your teams need to learn
- Leverage managed infrastructure as much as possible
- App Mesh is free, and the data plane proxies are very efficient
- A hop is a hop; keep your friends close, and your enemies closer

Relative simplicity

- Learn a handful of App Mesh constructs
- Learn the same number of Custom Resources
- App Mesh fades into the background
- Unlock all of the features

  - Observability
  - Security
  - Resilience
  - Traffic Management
  - More on the way!

For example

- Automatically inject sidecars and required environment for Jaeger, Prometheus, AWS X-Ray, and many other observability solutions
- Consistently implement retries and timeouts for your services, monitor for outlier behavior and configure circuit breakers
- Automate TLS termination for your service endpoints, include automated certificate management with AWS Certificate Manager
- Express your service-to-service traffic routing configuration

Traffic shifting: Adapt the pattern

With the simple traffic shifting building block in App Mesh, create

- ✅ Rolling Deployments
- ✅ Blue / Green Deployments
- ✅ Canary Releases
- ✅ A/B Testing Environments

We’re actively adding more features and capabilities, expanding the flexibility you have in how creative you can be

Go forth and build deployments

- Version your deployments
- Specify your App Mesh constructs
- Establish traffic flow configuration
- Leverage declarative configuration
- Automation is critical. More on that in a moment…

About those metrics

Metrics are central to automating incremental deployments

- Begin rollout
- Monitor metrics
- Continue or rollback as needed
- Rinse and repeat … however

Own less code

Flagger is a progressive delivery operator for Kubernetes

Uses App Mesh to automate analysis, testing, promotion and rollback of

- Canary Releases
- Blue / Green Deployments
- A/B Testing

Open source and maintained by Weaveworks

```
apiVersion: flagger.app/v1beta1
kind: Canary
metadata:
  name: podinfo
  namespace: test
spec:
  provider: appmesh:v1beta2
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: podinfo
  progressDeadlineSeconds: 60
  autoscalerRef:
    apiVersion: autoscaling/v2beta1
    kind: HorizontalPodAutoscaler
    name: podinfo
..
```

Own less code

Flux is the GitOps operator for Kubernetes

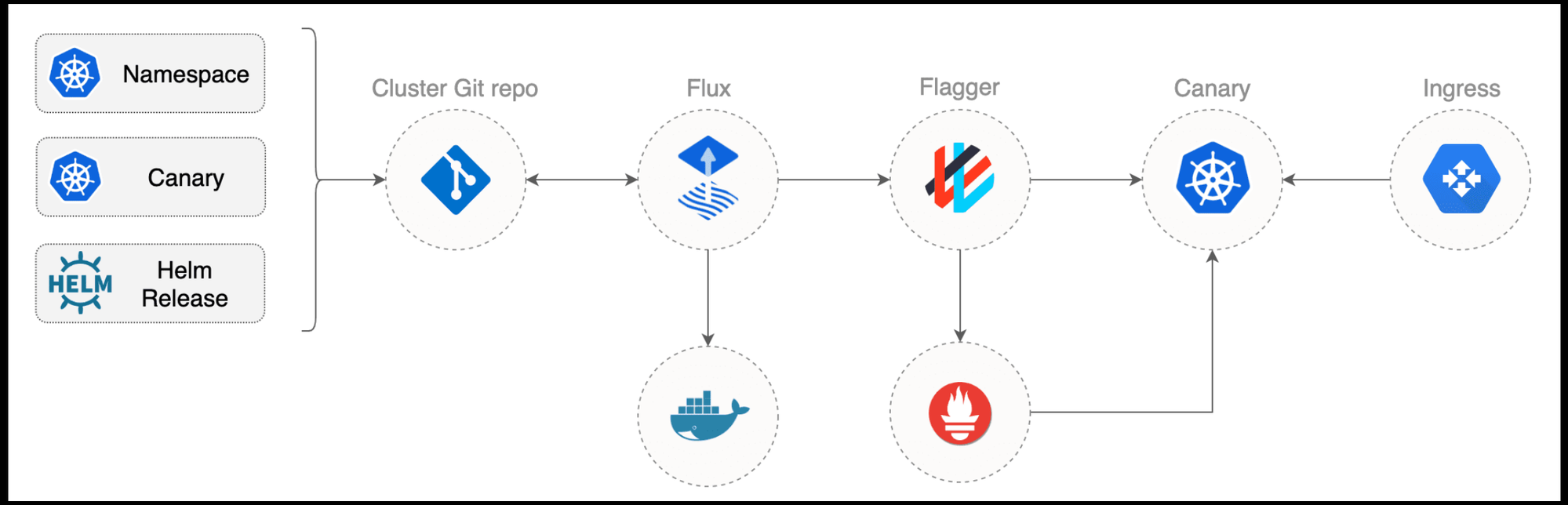
Triggers deployments from within your cluster, ensuring state matches Git

- Declarative
- Automated
- Focus on code, not containers



Open source and maintained by Weaveworks

Own less code



AWS Controllers for Kubernetes

A new open source project built with ❤️ by AWS

- Enables configuration and use of AWS managed service resources directly from Kubernetes
- Extends Kubernetes API through Custom Resources and Controllers for each supported service
- Removes the need to define managed service resources outside of your clusters



https://github.com/aws/aws-controllers-k8s

AWS Controllers for Kubernetes

Available to use now in Developer Preview

AWS Services included so far…

- Amazon API Gateway V2
- Amazon DynamoDB
- Amazon ECR
- Amazon S3
- Amazon SQS
- Amazon SNS
- Amazon ElastiCache

```
apiVersion: "ecr.services.k8s.aws/v1alpha1"
kind: Repository
metadata:
    name: "my-ecr-repo"
spec:
    repositoryName: "encrypted-repo-managed-by-ack"
    encryptionConfiguration:
        encryptionType: AES256
    tags:
    - key: "is-encrypted"
      value: "true"
```

Next steps

A robust and solid system to automate change is table stakes

Teams are empowered to run fast at problems without worrying about babysitting deployments

- Fix issues as fast as possible, continuously deploying the latest
- Add new features without fear of blowing up the system
- Refactor and redesign components at will

Next steps

There are other ways to implement deployments, but

- ✅ Own less code
- ✅ Leverage the systems you have chosen
- ✅ Pass off undifferentiated heavy lifting to managed services

Next steps



Once you've invited App Mesh to the party

- Observability
- Security and policy
- Improved resilience

```
routes:
  - name: route-foo
    httpRoute:
      match:
        prefix: /foo
      action:
        weightedTargets:
          - virtualNodeRef:
              name: foo-prod
            weight: 1
      retryPolicy:
        maxRetries: 4
        perRetryTimeout:
          unit: ms
          value: 2000
        httpRetryEvents:
          - server-error
  - name: route-bar
    httpRoute:
      match:
        prefix: /bar
..
```

Connect with us

If you’re new to App Mesh, check out the GitHub repo of examples and walkthroughs

- https://github.com/aws/aws-app-mesh-examples

As always, our work is guided by you; visit the AWS Container Services and AWS App Mesh Roadmaps on GitHub

- https://github.com/aws/containers-roadmap
- https://github.com/aws/aws-app-mesh-roadmap

Thank you!

Jesse Butler

@jlb13

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.