INNOVATE

MODERN APPLICATIONS EDITION

27&28 October 2021

Architecting Kubernetes for seamless deployments and upgrades with Amazon EKS

Jason Umiker
Senior Specialist Solutions Architect, Containers
Amazon Web Services

aws

Agenda

Application considerations

- API and schema version compatibility
- Shutdown, probes, and new version scaling
- Dealing with state
- Dealing with dependencies

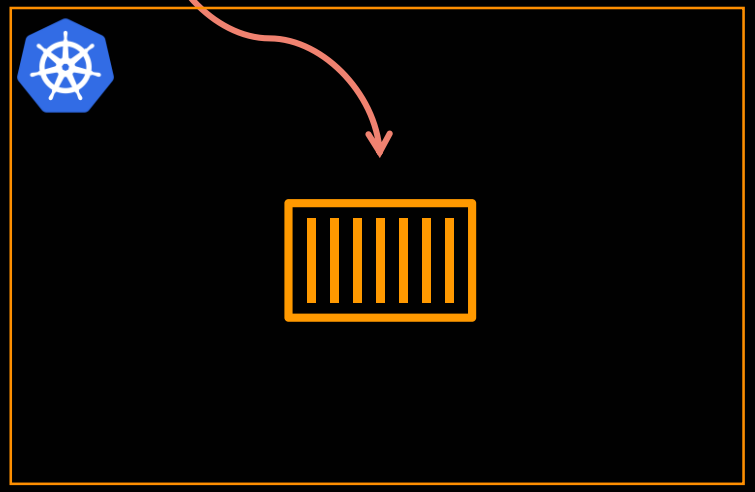
Cluster considerations

- Any node upgrade = Instance replacement
- Kubernetes API deprecations
- Node group upgrade = Replacement of all pods
- Kubernetes upgrades have flow-on upgrades

The inspiration for this talk

TO DISPEL THE MYTH THAT

Unchanged legacy app



= “All of my problems are solved”

Why are we starting with application updates?

- The cluster is like this roundabout with the cars as the Services/Pods:

- It can be very smooth and efficient - but only if all of the individual cars driving on it ‘follow the rules’ and behave as expected collectively.

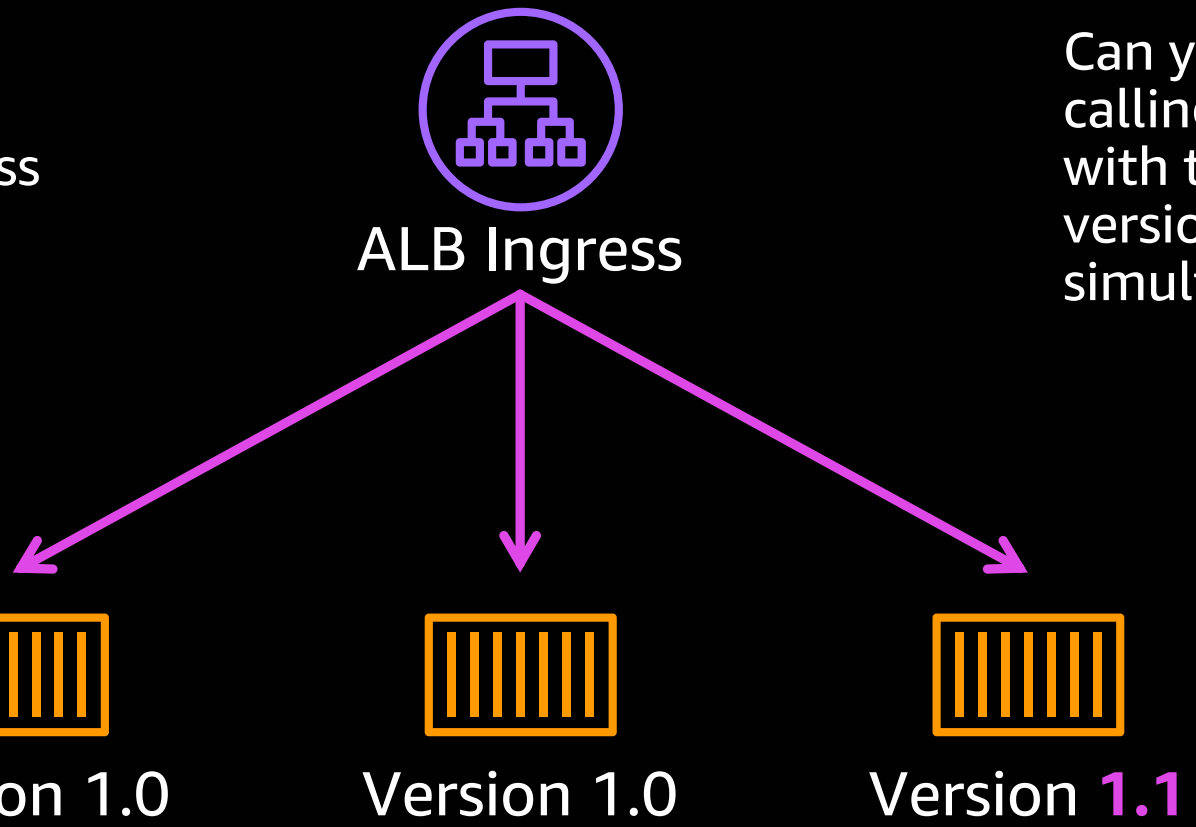
Application considerations

API and schema version compatibility

API and schema version compatibility

KUBERNETES DEPLOYMENTS = ROLLING UPDATES

Both new and old versions will be behind the same service/ingress during deployment



Can your front-end or calling microservice deal with talking to two versions of the service simultaneously?

API and schema version compatibility

ROLLING UPDATE = API BACKWARDS COMPATIBILITY

If the upstream client or front-end needs to upgrade/deploy at the same time as your service this is referred to as a tight coupling or a distributed monolith.

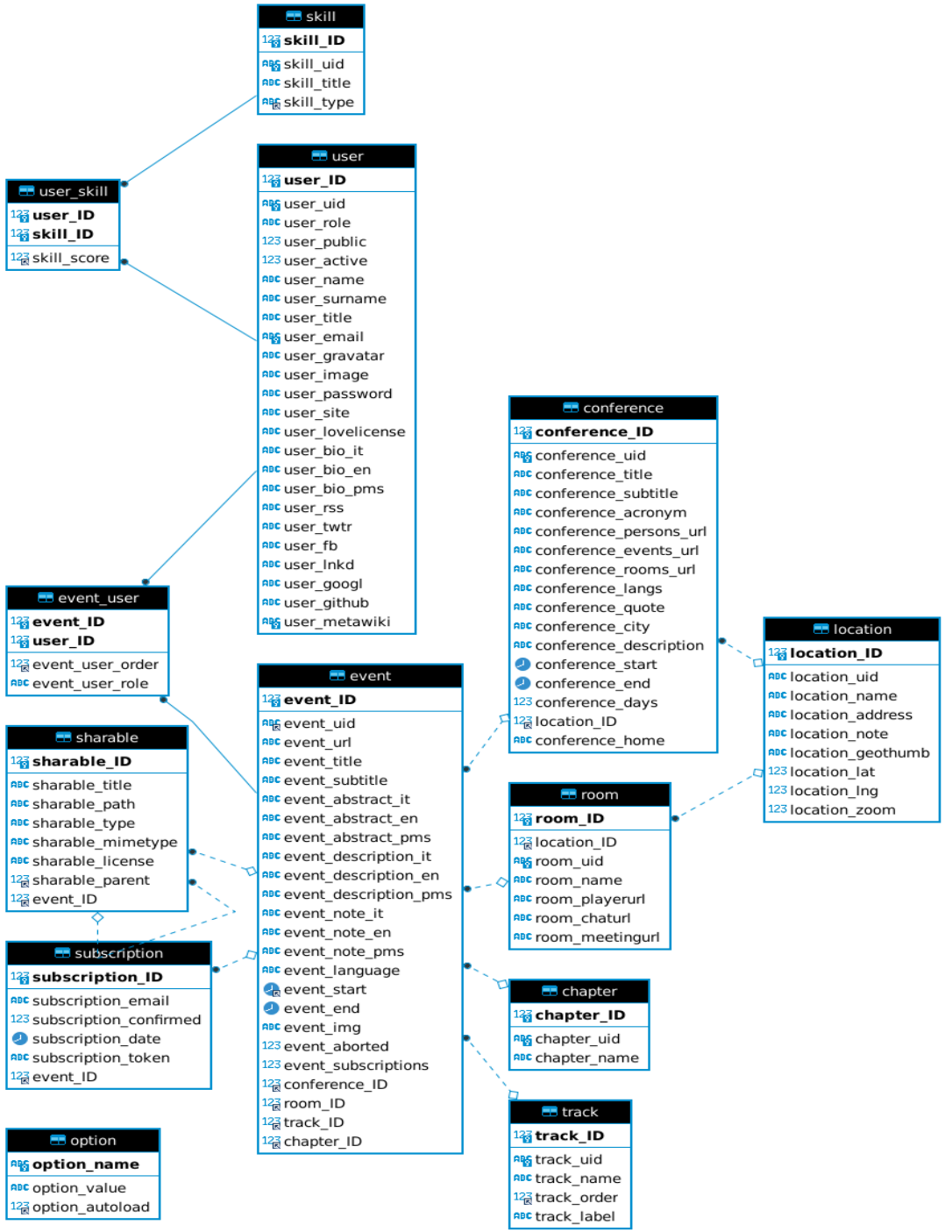
Microservices are about decoupling services and teams so they can deploy independently – and ideally without having to talk to each other!

Can you support n-1 API version today?

API and schema version compatibility

DATABASE SCHEMA CREATION/UPDATES?

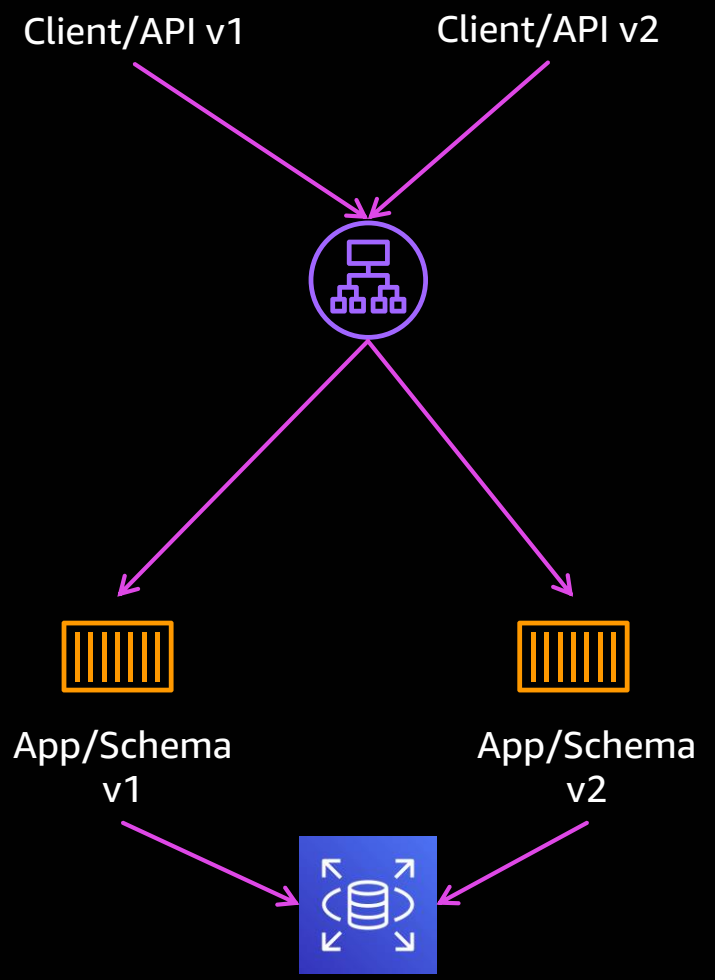
- How do you do your database schema updates today?
- If it's a manual pre-deployment runbook of running scripts with .sql files then is there a better way?
- Can Kubernetes help here?



Architecting for rolling updates

- **In order to cope with the rolling updates:**
- Support the current API version and at least (n-1)

- Manage your database schema changes such that:
  - You version your schema and track changes
  - You have upgrade logic/steps for each version upgrade
  - You limit ALTER operations to non-blocking ones (e.g. add new column rather than update column)
  - You have specific SELECTs (i.e. not * so if a new column appears the old version will get the same response)

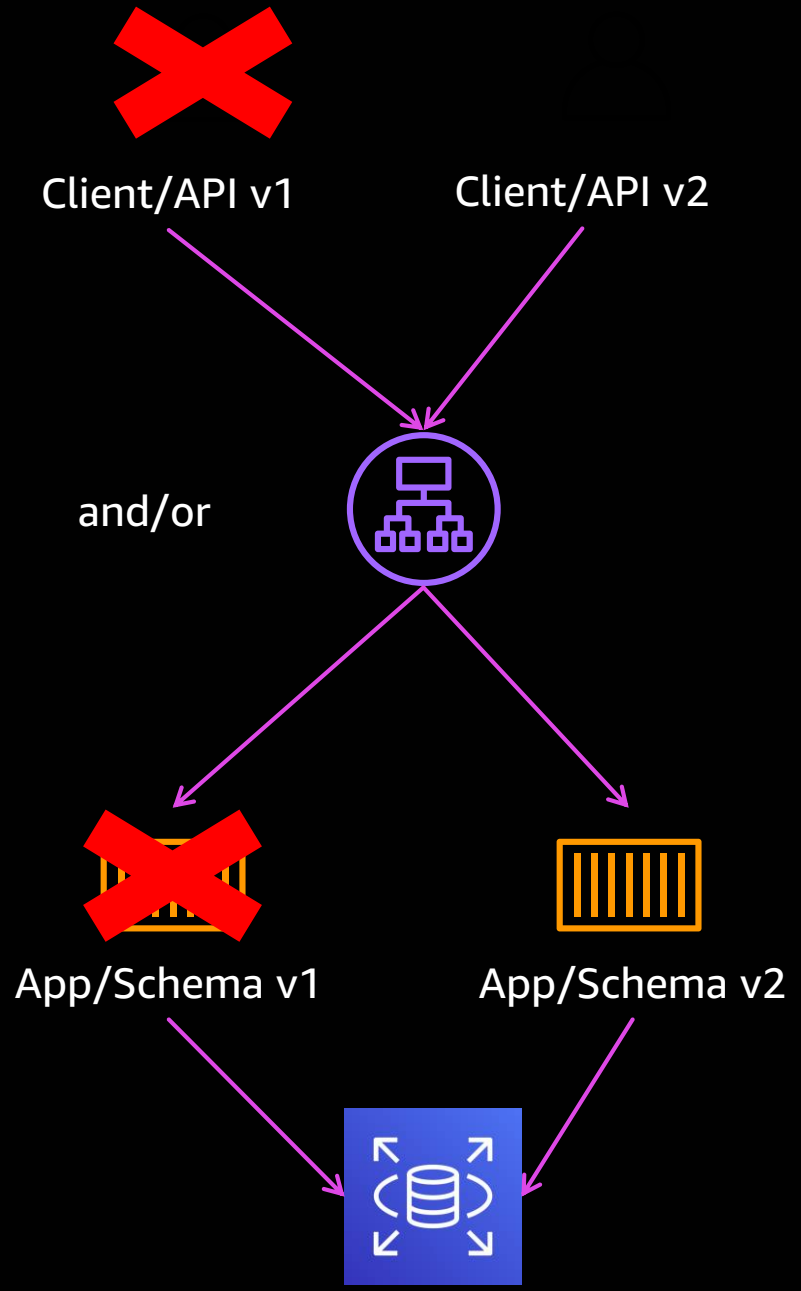


Blue/Green for when you can't

- Alternatively, you can do a blue/green deployment whereby two versions of the app will never be running at the same time

- You can do this to:
- Protect from incompatible clients
- Protect from incompatible database schema code
- or both

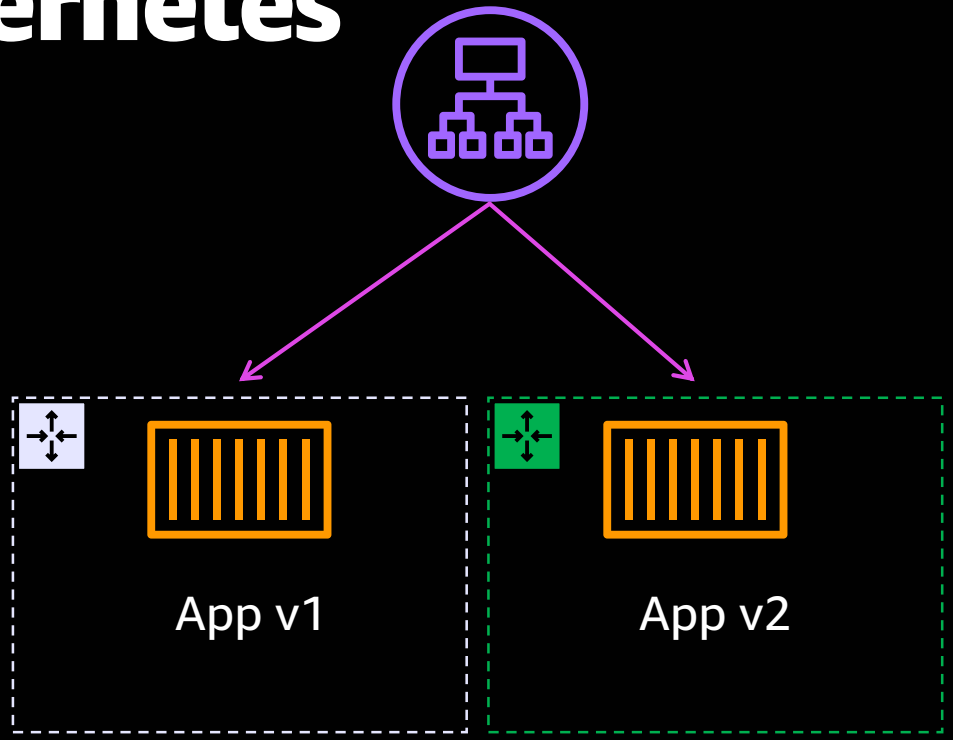
- The tradeoff is that deployments to update or rollback will have an (often brief) disruption/outage



How to do blue/green on Kubernetes

- Kubernetes deployments do not support blue/ green themselves but you can accomplish one by:

1. Creating a new second deployment with the new green version
2. Testing to ensure that the green version came up successfully
3. Changing the selector on the service/ingress that fronts the application pods to flip 100% of the traffic to the new green deployment
4. Scale down the blue deployment

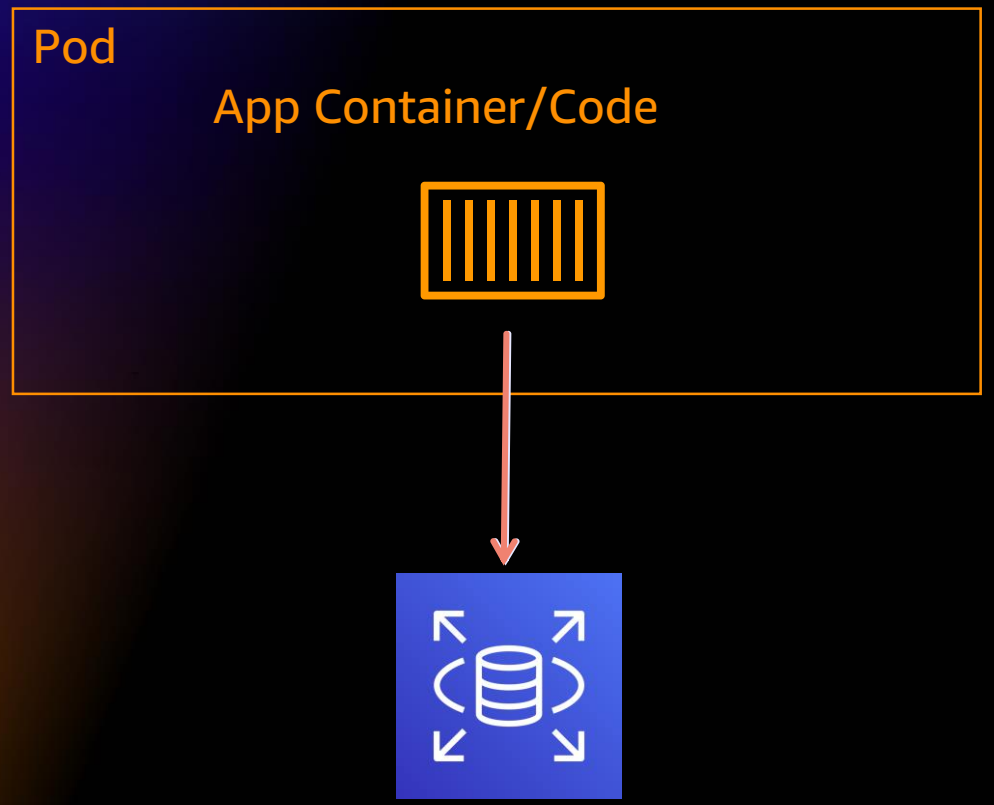



```
kind: Service
metadata:
  name: app
spec:
  selector:
    app: app-v2
```

Schema updates via Kubernetes init containers?

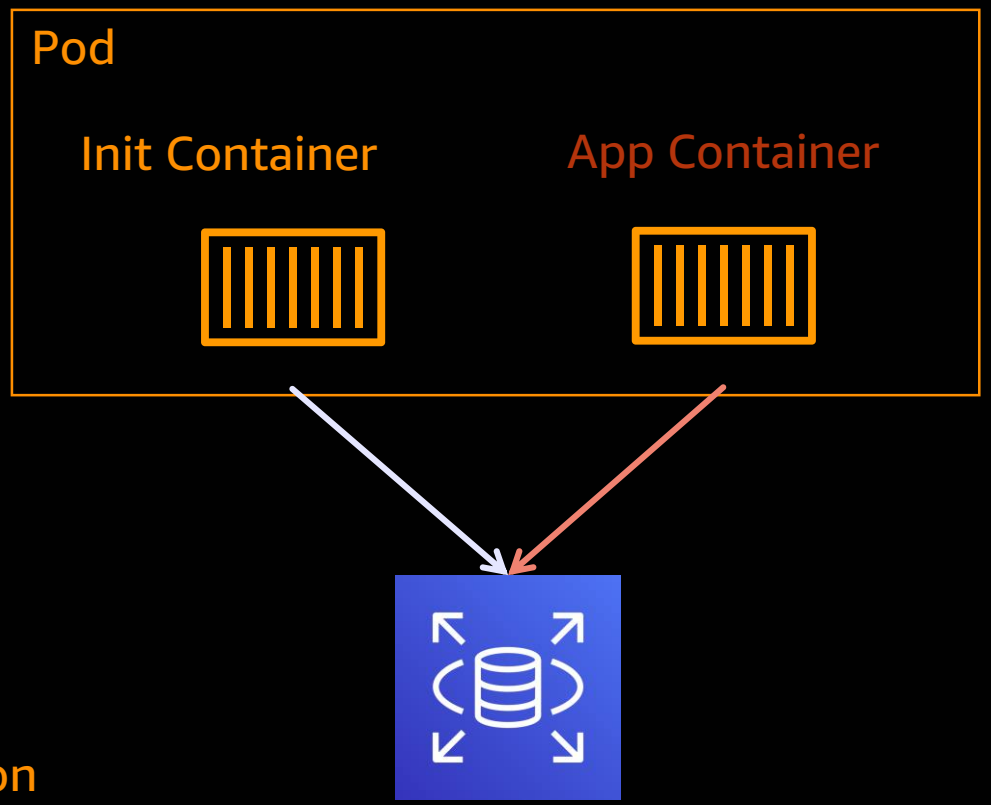
Two of the common approaches to database schema updates are:

Coding it within your app:



vs.

Scripting it in an init container:



Shutdown, probes, and new version scaling

Shutdown, probes, and new version scaling

DOES YOUR APPLICATION HONOUR SIGTERM OR NEED LONGER THAN 30S TO SHUT DOWN?

Kubernetes wants your pod to go away (Node drain, ReplicaSet scale in, etc.)

It issues a **SIGTERM** to each container in the Pod instructing it to gracefully wrap up and shut itself down

It waits up to 30s for this to happen by default* then issues a **SIGKILL**

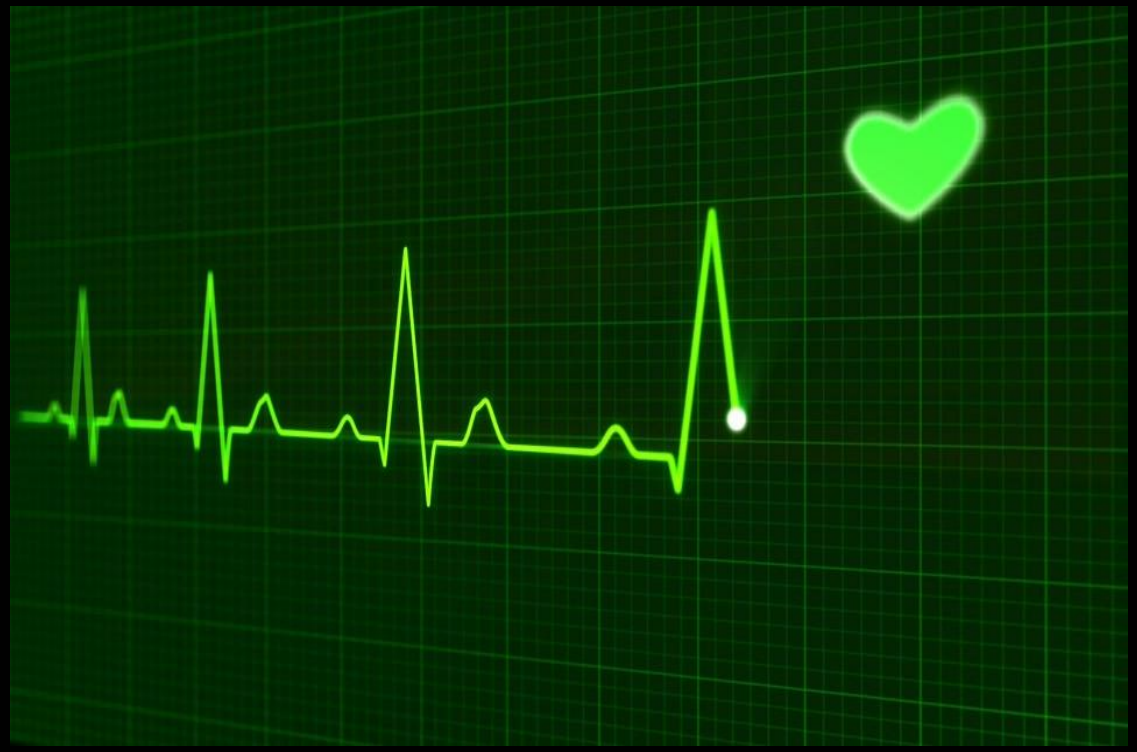
Shutdown, probes, and new version scaling

Do you have the right Readiness, Liveness and Startup Probes?

Readiness
If it fails stop sending it traffic

Liveness
If it fails kill/heal it

Startup
Don’t start the readiness and liveness probes until this passes

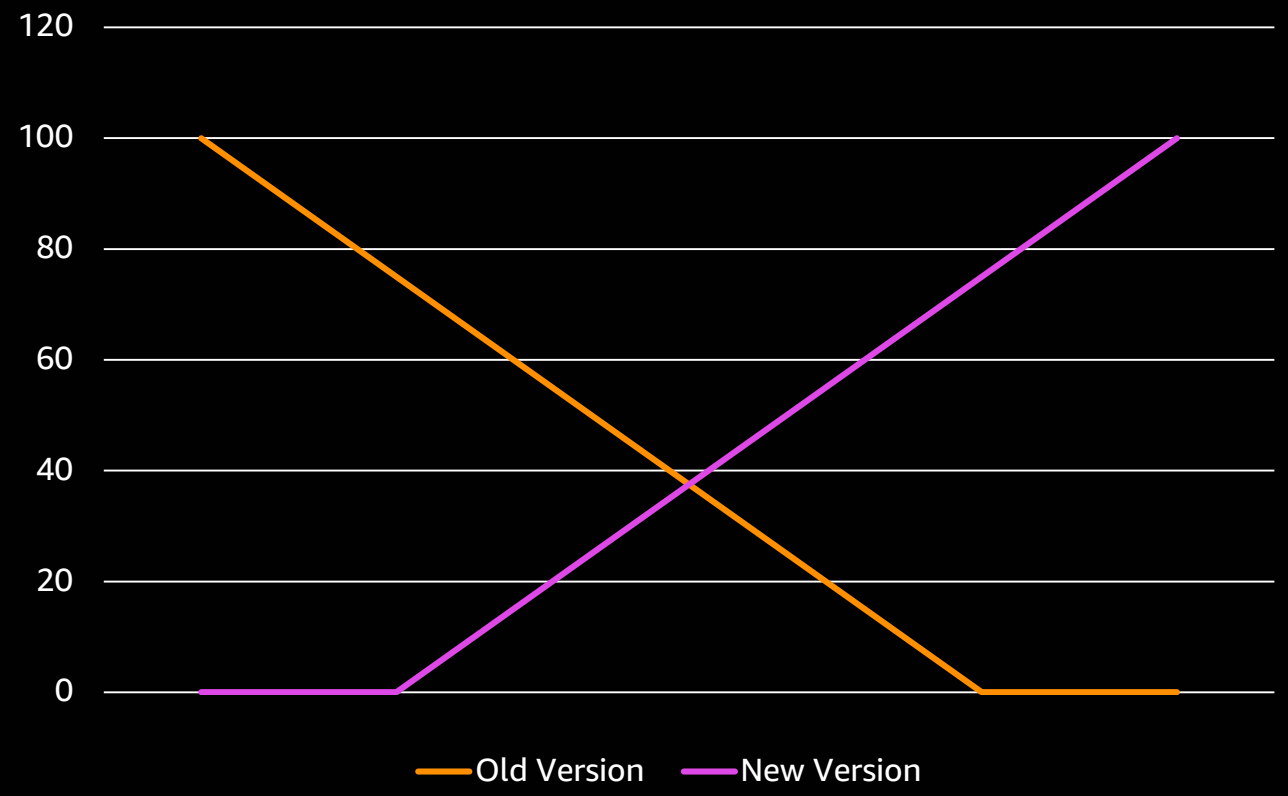


Shutdown, probes, and new version scaling

CAN YOU COPE WITH 75% OF YOUR CAPACITY?

By default, a Deployment ensures you don't consume more resources during its rolling update process.

It starts by scaling down the old version by 25% *before* scaling up the new by 25% (leaving a gap especially if the new version doesn't work).



How much capacity during deployments?

How you control this depends on whether you are doing a rolling update or not:

Deployment rolling-update:

You can tweak the `maxUnavailable` and `maxSurge` parameters on the Deployment to allow for more capacity through the process.

Blue/Green:

You are spinning up a second deployment and thus control the scaling of both deployments independently

Code a graceful shutdown on SIGTERM

If you don't handle SIGTERM then it usually means that any callback that's pending, any network request still being sent, any filesystem access, or processes writing to STDOUT or STDERR - all is going to be ungracefully terminated right away!

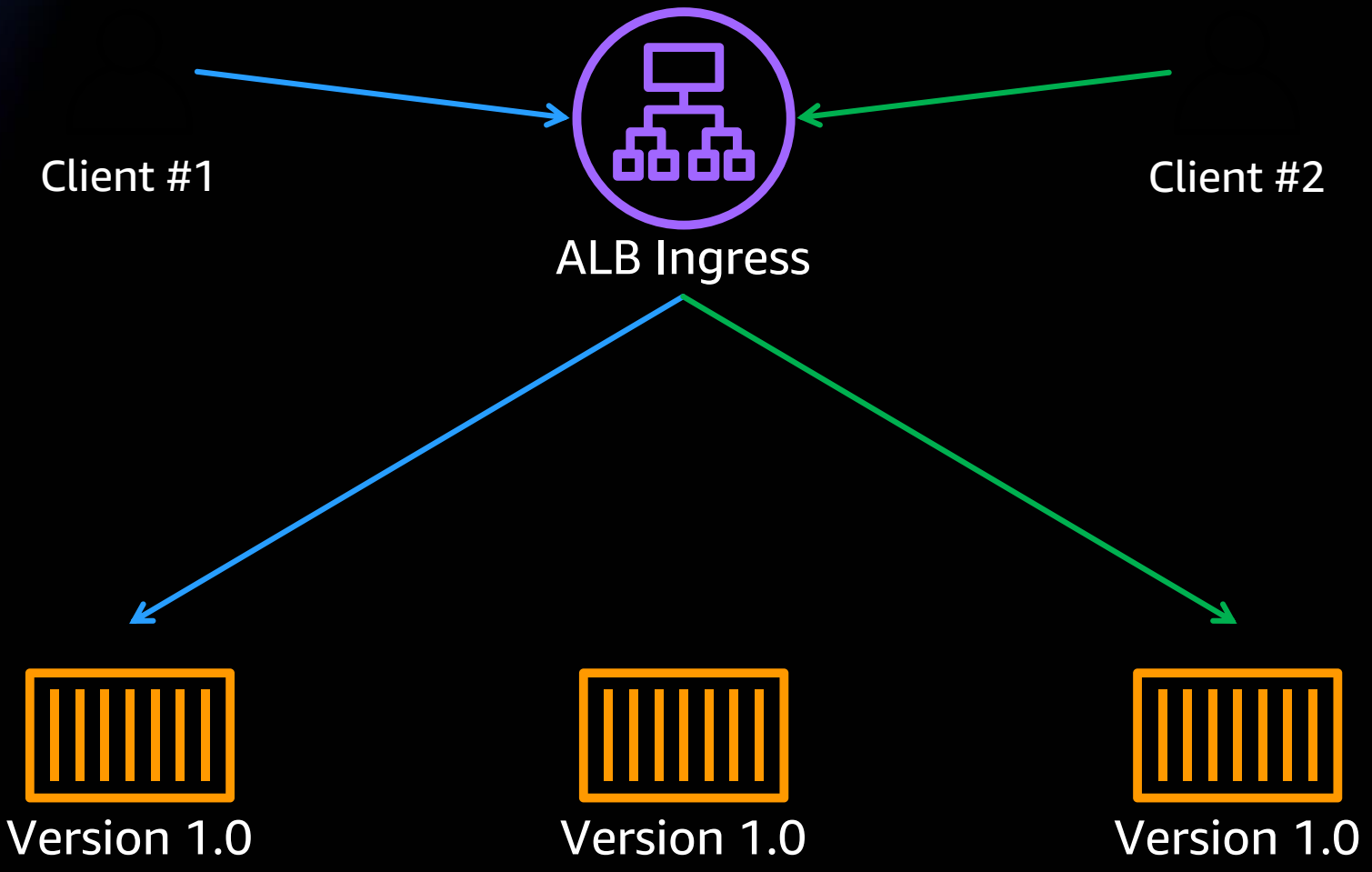
And time it to ensure the terminationGracePeriodSeconds is long enough (defaults to 30 seconds)…

```
process.on('SIGTERM', () => {
  console.info('SIGTERM signal received.');
  console.log('Closing http server.');
  server.close(() => {
    // Close DB connections, other chores, etc.
    });
  });
```

Dealing with state

Dealing with state

DO YOU HAVE IN-MEMORY STATE REQUIRING THE CLIENTS TO GET SERVED BY THE SAME PODS?

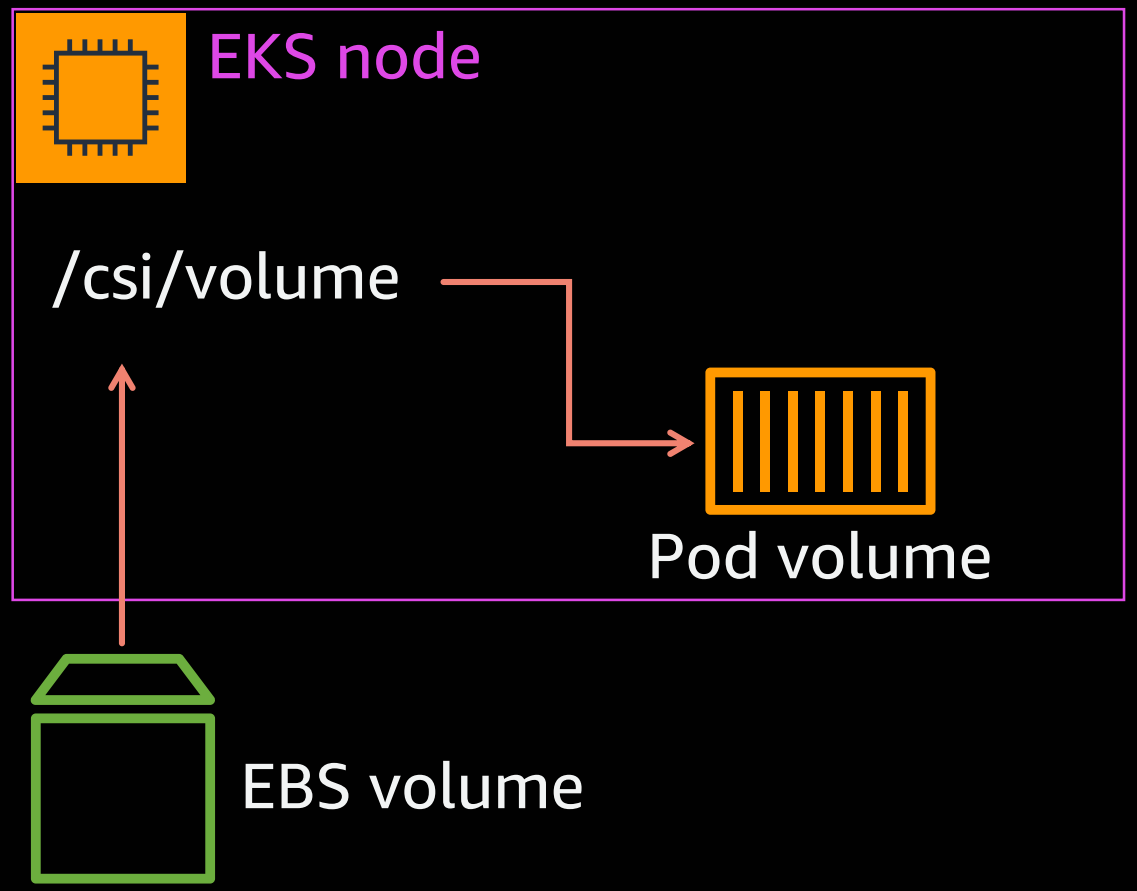


Dealing with state

- Do you need to store state in a disk that persists as the pod comes and goes?

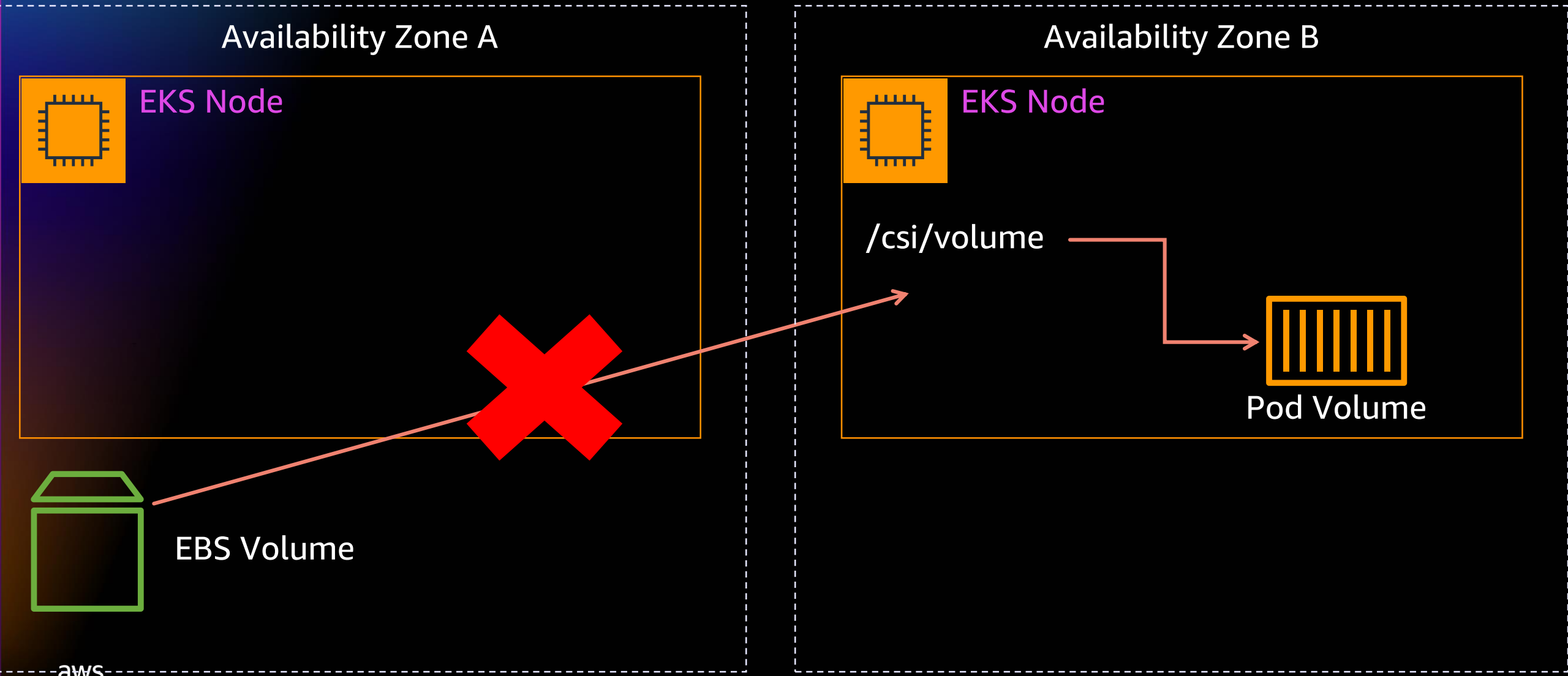
- When you use the AWS EBS type of PersistentVolume it can only be mounted to one EC2 Instance / EKS node at a time

- As such, there is a brief outage if it needs to be rescheduled onto a new node for the pod to stop and the volume to be unmounted before it can be remounted



Dealing with state

- Do you need to store state in a disk that persists as the pod comes and goes?



How to do sticky sessions

- Kubernetes' built-in service is limited here but Ingress(es) open up more options:

Service session affinity

This will do a sticky session based on the source IP address. Its only option is `timeoutSeconds` which defaults to 3 hours.

Ingress

When using an Ingress you can configure sticky sessions via annotations. For example, you can do a cookie based stickiness with the AWS LB Controller with:

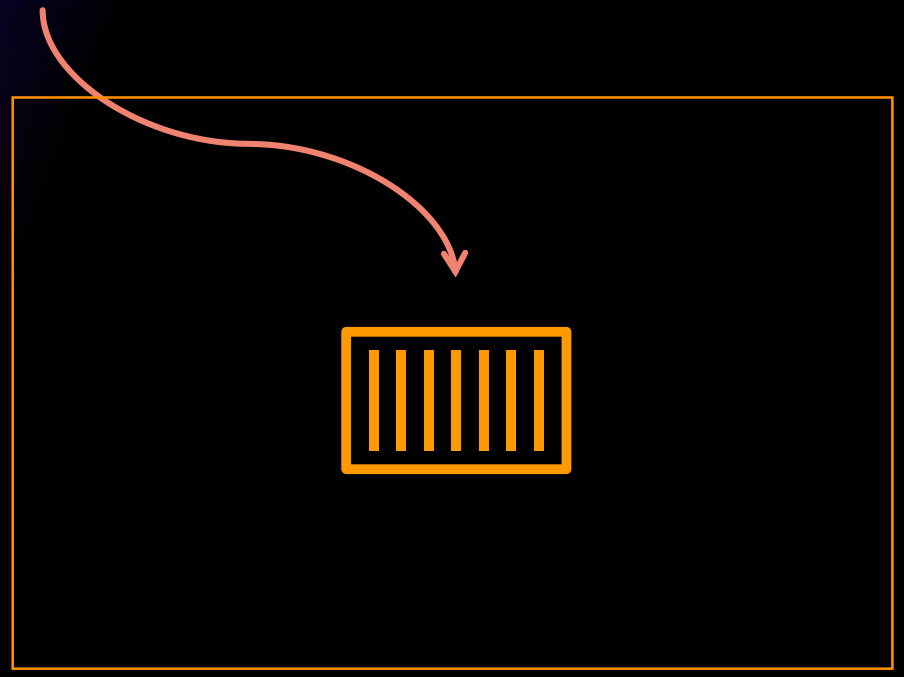
```
alb.ingress.kubernetes.io/target-group-attributes:
stickiness.enabled=true,
stickiness.lb_cookie.duration_seconds=60
```

To sticky or not to sticky?

- Just because you can doesn't mean that you should:

Pros

Unchanged legacy app



Cons

Any healing of pods, deployment of changes, or cluster upgrades *may* have user-facing disruption and so you *may* have to do them during outage windows.

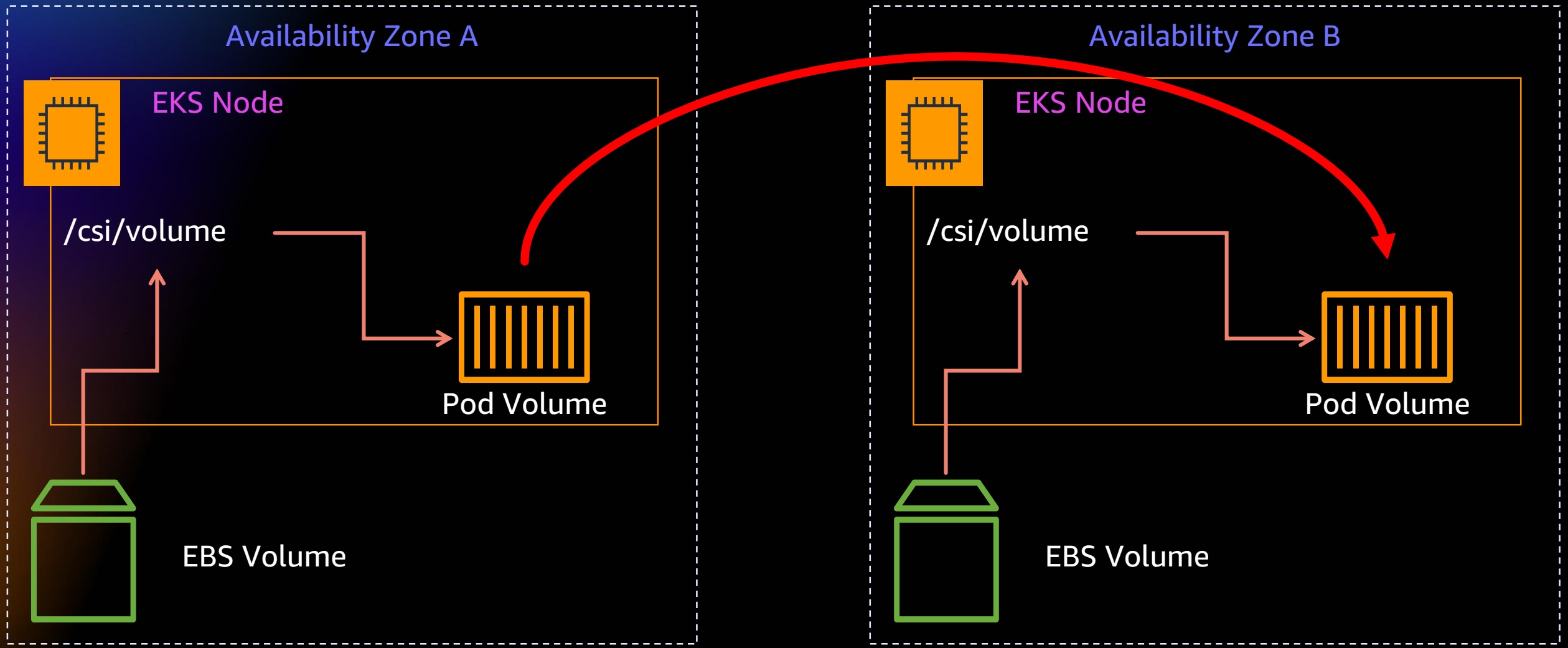
Which type of PersistentVolume?

- AWS's has three types of CSI drivers:

Storage Type	Dynamic Provisioning?	Single or Multi-AZ?	Single or Multi-Mount?
EBS	Yes	Single-AZ	Single-Mount
EFS	Yes	Multi-AZ	Multi-Mount
FSx Lustre	Yes	Single-AZ	Multi-Mount

EBS PersistentVolumes w/o disruption?

IF THE APP REPLICATES ITS STATE TO ANOTHER POD WITH ITS OWN PERSISTENTVOLUME

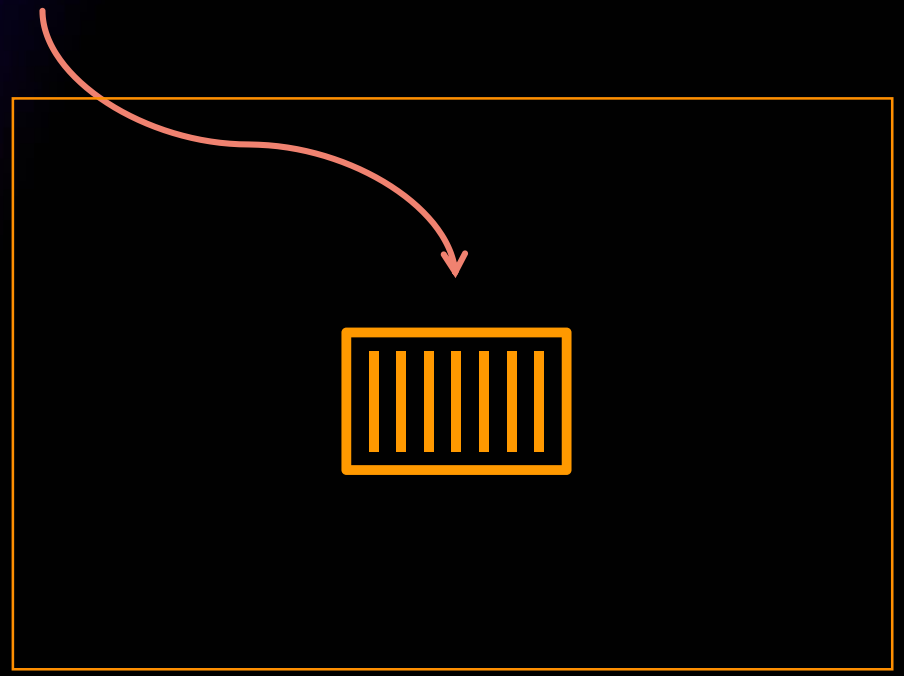


To PersistentVolume or not to PersistentVolume

Just because you can doesn't mean that you should:

Pros

Unchanged legacy app



Cons

Any healing of pods, deployment of changes, or cluster upgrades *may* have user-facing disruption (EBS mainly) and so you *may* have to do them during outage windows.

Dealing with dependencies

Dealing with dependencies

DEPENDENCIES BETWEEN CONTAINERS / PODS?

- When building a bridge you need to finish the arch before you can hang the roadway under it.

- Similarly, do the containers within your pod, or the pods within your service, have a particular order or dependencies that is required on startup?

Deployments vs StatefulSets

FOR WHEN ALL PODS ARE NOT CREATED EQUAL

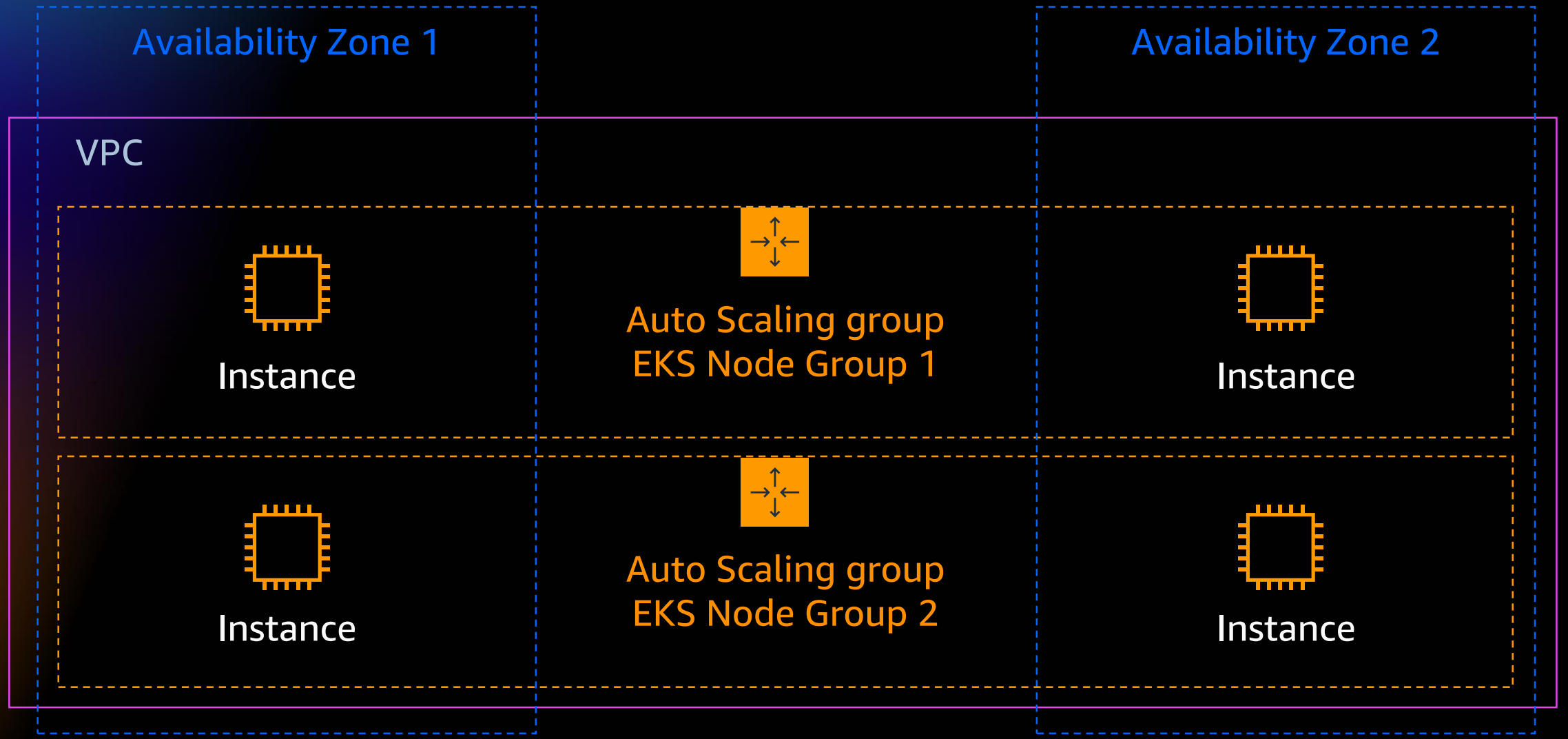
StatefulSets by default are:

- Created one at a time in order
- Deleted one at a time in the opposite order (last to first)
- Updated in a rolling update in the opposite order (last to first)
- Given stable names that follow them in the format of [name]-[incrementing number] e.g. app-0, app-1
- PersistentVolumeClaims go with the pod(s) when they are rescheduled on another node

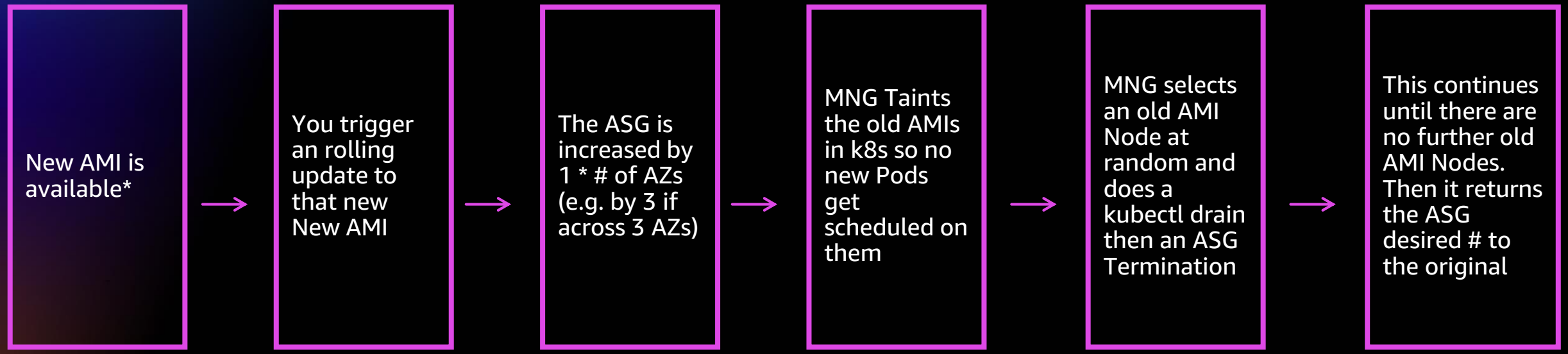
Cluster considerations

Any node upgrade = instance replacement

WORKER NODES ARE IN EC2 ASGS AND ARE IMMUTABLE



Managed node groups help make this seamless

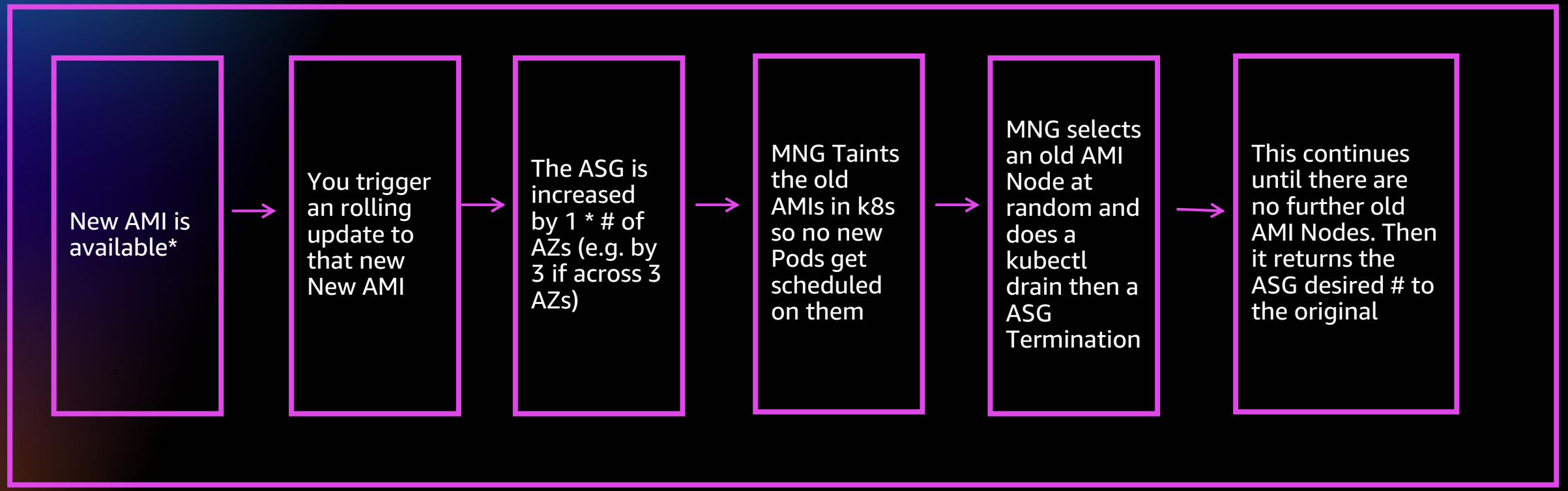


*This is either because there is a new update for your existing Kube version or you just upgraded Kube to a new version and now need to update the Nodes too

Managed node groups help make this seamless

You have to do all this yourself in self-managed

AWS Fargate helps make this even more seamless



You don't have to do any of this with AWS Fargate

Node group upgrade = replacement of all pods

WORKER NODES ARE IN EC2 ASGS AND ARE IMMUTABLE

Pod disruption budgets

Limit how much particular services can be disrupted during the process

```yaml
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: zk-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: zookeeper
```

Kubernetes 1.5+

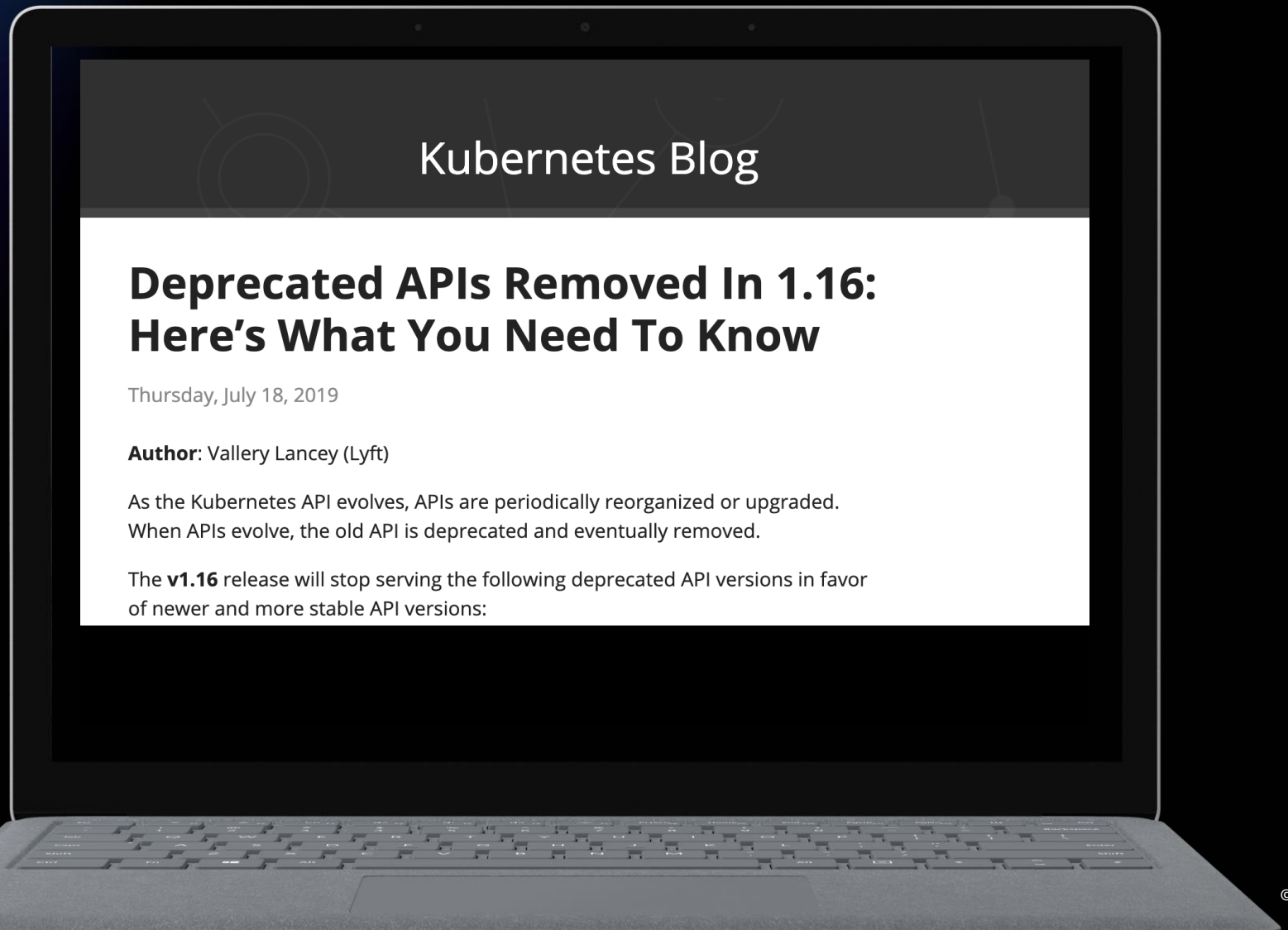
OR

```yaml
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: zk-pdb
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: zookeeper
```

Kubernetes 1.7+

Kubernetes API deprecations

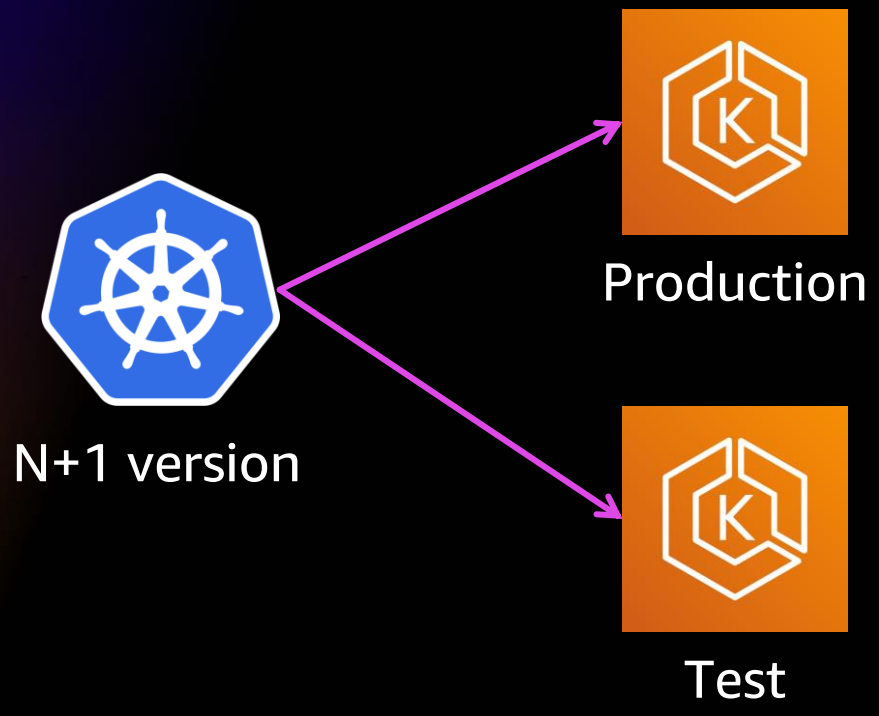
SOMETIMES APIS HAVE BEEN DEPRECATED AND YOUR YAML SPECS MUST BE CHANGED BEFORE THE UPGRADE



Testing vs scanning your YAML re: deprecations

Trying it is the only way to be sure – but scanning can help be confident

Testing



Scanning/linting

There are various tools on GitHub such as:

- Deprek8ion - can scan your YAML ad-hoc or in your pipeline and warn or error out
- kube-no-trouble – can scan your cluster(s) and tell you if it contain deprecated YAML

Kubernetes upgrades have flow-on upgrades

OFTEN REQUIRED ADD-ONS NEED TO BE UPGRADED TOO

Kubernetes version	1.18	1.17	1.16	1.15
Amazon VPC CNI plug-in	1.7.5	1.7.5	1.7.5	1.7.5
DNS (CoreDNS)	1.7.0	1.6.6	1.6.6	1.6.6
KubeProxy	1.18.9	1.17.12	1.16.15	1.15.12

Managed cluster add-ons (will) help here

Let us manage the infrastructure Pods so you can focus on your Pods

- Add-ons we ship with EKS today but you manage after creation:
  - CNI (the first we're doing as a Managed Add-on!)
  - kube-proxy
  - CoreDNS

- Others you may install/need:
- Fluentbit (log shipping), cluster and horizontal pod auto-scalers, Ingress controller, OPA Gatekeeper, etc.

Separating the myth from the reality

YOUR CHOICES W/STATE & CLIENT STICKINESS CAN MAKE DEPLOYMENTS/UPGRADES DISRUPTIVE

To recap:

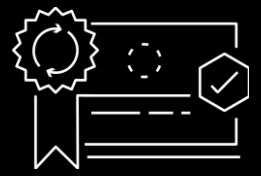
- Kubernetes doesn't magically solve your legacy app challenges!
- If you can't architect for rolling updates then you can do blue/green.
  - But then deployments and rollbacks may be disruptive.
- If you can't architect for statelessness (i.e. storing your state in external databases, caches, queues, etc.) then you can use sticky sessions and/or PersistentVolumes.
  - But then both deployments and cluster upgrades may be disruptive.
- There are Kubernetes upgrades every quarter – so you need to either architect to make them seamless or negotiate the outage windows to allow for them.

Happy Kubernetes-ing on AWS!

AWS Training and Certification

Accelerate modernization with continuous learning

Free digital courses, including:
Architecting serverless solutions
Getting started with DevOps on AWS



Earn an industry-recognized credential:
AWS Certified Developer – Associate
AWS Certified DevOps – Professional

Hands-on classroom training (available virtually) including:
Running containers on Amazon Elastic Kubernetes Service (Amazon EKS)
Advanced developing on AWS

Create a self-paced learning roadmap
AWS ramp-up guide - Developer
AWS ramp-up guide - DevOps

Take Developer and DevOps training today

Learn more about Modernization training for you and your team

Visit the Modern Applications Resource Hub for more resources

Dive deeper with these resources to help you develop an effective plan for your modernization journey.

- Build modern applications on AWS e-book
- Build mobile and web apps faster e-book
- Modernize today with containers on AWS e-book
- Adopting a modern Dev+Ops model e-book
- Modern apps need modern ops e-book
- Determining the total cost of ownership: Comparing Serverless and Server-based technologies paper
- Continuous learning, continuous modernization e-book
- … and more!

https://bit.ly/3yfOvbK

Visit resource hub »

Thank you for attending AWS Innovate Modern Applications Edition

We hope you found it interesting! A kind reminder to **complete the survey.**
Let us know what you thought of today's event and how we can improve the event experience for you in the future.


aws-apj-marketing@amazon.com


twitter.com/AWSCloud


facebook.com/AmazonWebServices


youtube.com/user/AmazonWebServices


slideshare.net/AmazonWebServices


twitch.tv/aws

Thank you!