KubeCon CloudNativeCon North America 2025

PROCORE

Zero Downtime Migration of Monolith from ASG to K8s

using Sidecar and Container Lifecycle Hooks

James Dabbs jcdabbs

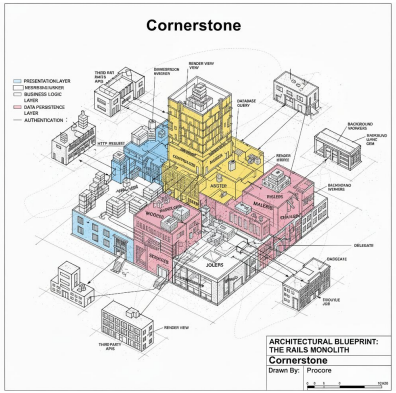
Deepak Kosaraju deepak-kosaraju

#KubeCon #CloudNativeCon

Agenda

KubeCon CloudNativeCon North America 2025

Goal: trace Procore's path to Kubernetes - bumps, learnings and all



Background

Understand the state we were migrating from, what we hoped to get from the migration, and the constraints we'd have to engineer for.

Features

Survey the tools at our disposal - K8s primitives and Envoy - and how we deployed them to migrate smoothly and safely.

Demo

Demonstrate with live examples of how a system behaves when using those tools in different configurations.

Procore & Cornerstone

"Building the software that builds the world"

- **Cornerstone** - our legacy Rails monolith - started c. 2007 at Rails version 0.8.7
- ~90 teams contributing ~33 PRs/day
- Near-continuous deploys, ~9/day
- Handles ~10K heterogenous requests per second

Procore & Cornerstone

All happy microservices are alike;
each unhappy monolith is unhappy in its own way

– Tolstoy, *Anna Karenina (1878)*

The Cornerstone Monolith

“Auto Scaling” Groups

~450 nodes (`m5.2xlarge`ish) across regions

Challenges

- Homegrown compute orchestration automation that was difficult to ramp up on, reason about, or change safely
- Legacy debt accrued downstream of 👆
- Overprovisioning, exacerbated by near-continuous blue-green deployment
- Configuration drift between environments

The Cornerstone Monolith

Strengths

- Good internal Kubernetes expertise
- Current architecture is pod-like
- Executive support for foundational rework

Weaknesses

- App is far from cloud native - especially w.r.t. startup, shutdown, and state
- Unproven experience at this scale or criticality

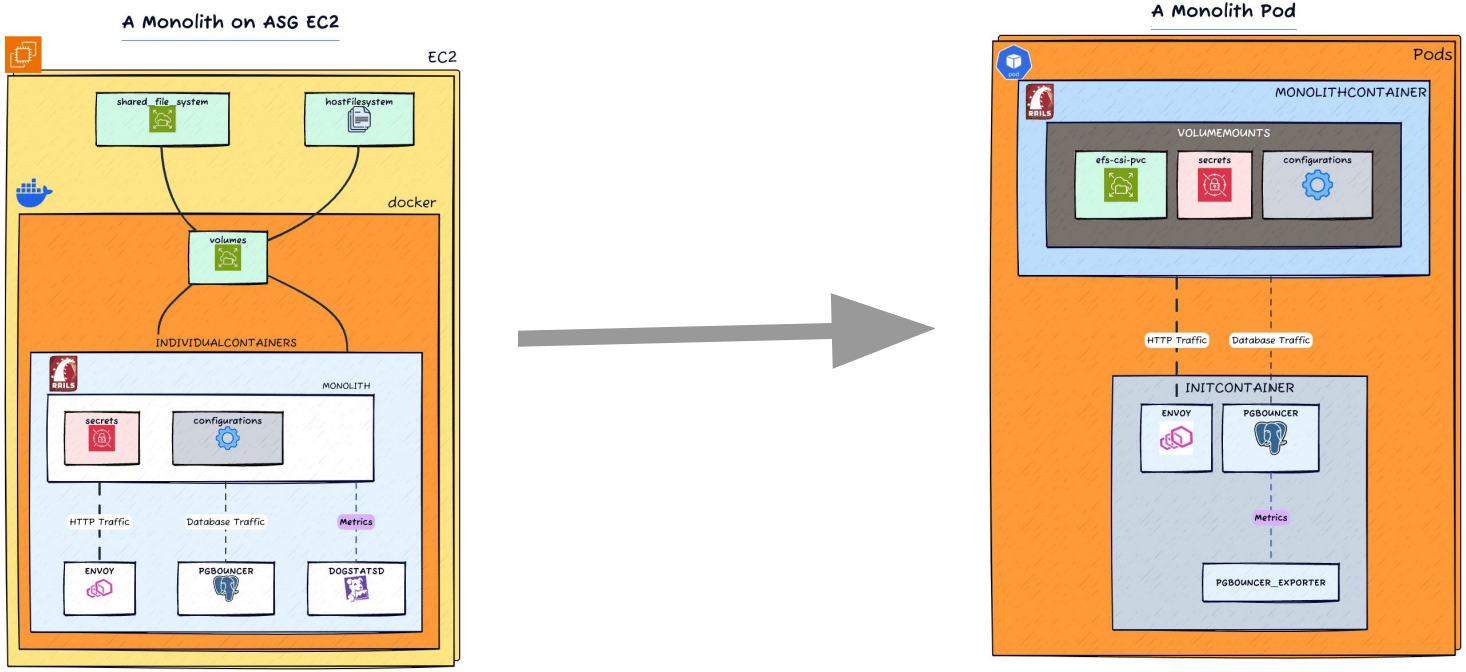
Opportunities

- Open the application's black box
- Evolve rollout-and-back safe development practices

Threats

- Executive support may not weather high-profile outages, cost overruns, languishing in development

The Cornerstone Monolith



The Cornerstone Monolith

Strengths

- Good internal Kubernetes expertise
- Current architecture is pod-like
- Executive support for foundational rework

Weaknesses

- App is far from cloud native - especially w.r.t. startup, shutdown, and state
- Unproven experience at this scale or criticality

Opportunities

- Open the application's black box
- Evolve rollout-and-back safe development practices

Threats

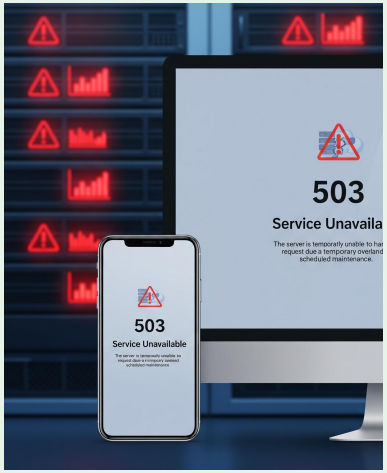
- Executive support may not weather high-profile outages, cost overruns, languishing in development

Cornerstone on Kubernetes

Hypothesis: by migrating to Kubernetes

We can tighten our scaling behavior without anyone* noticing

* Internal feature teams or external customers



Cornerstone on Kubernetes Timeline

KubeCon CloudNativeCon North America 2025

Feb ‘24
Cornerstone Offsite

Greenlight the migration effort and start design and planning work in earnest.

July ‘24
Initial Deployment & Experiment

Bootstrap EKS clusters, build migration pipelines, experiment with Karpenter for node provisioning, and initiate a **1%-5%** traffic shift of critical services.

Aug - Sept ‘24
Rolling Traffic Migration

Scale traffic shift from **15% to 100%**. Introduce Horizontal Pod Autoscaling (HPA) with both scheduled and demand-based scaling to optimize resource utilization.

Oct ‘24
ASG Sunset & Advanced Autoscaling

Complete legacy Auto Scaling Group sunset.

Refined demand-based HPA scaling using KEDA.

Q1 ‘25
Full Demand-Based Scaling & Canary Rollouts

Achieve **100%** demand-based HPA scaling across all services.

Start dark rollout of canary deployments.

Optimize Karpenter binpacking and deployment times for efficiency.

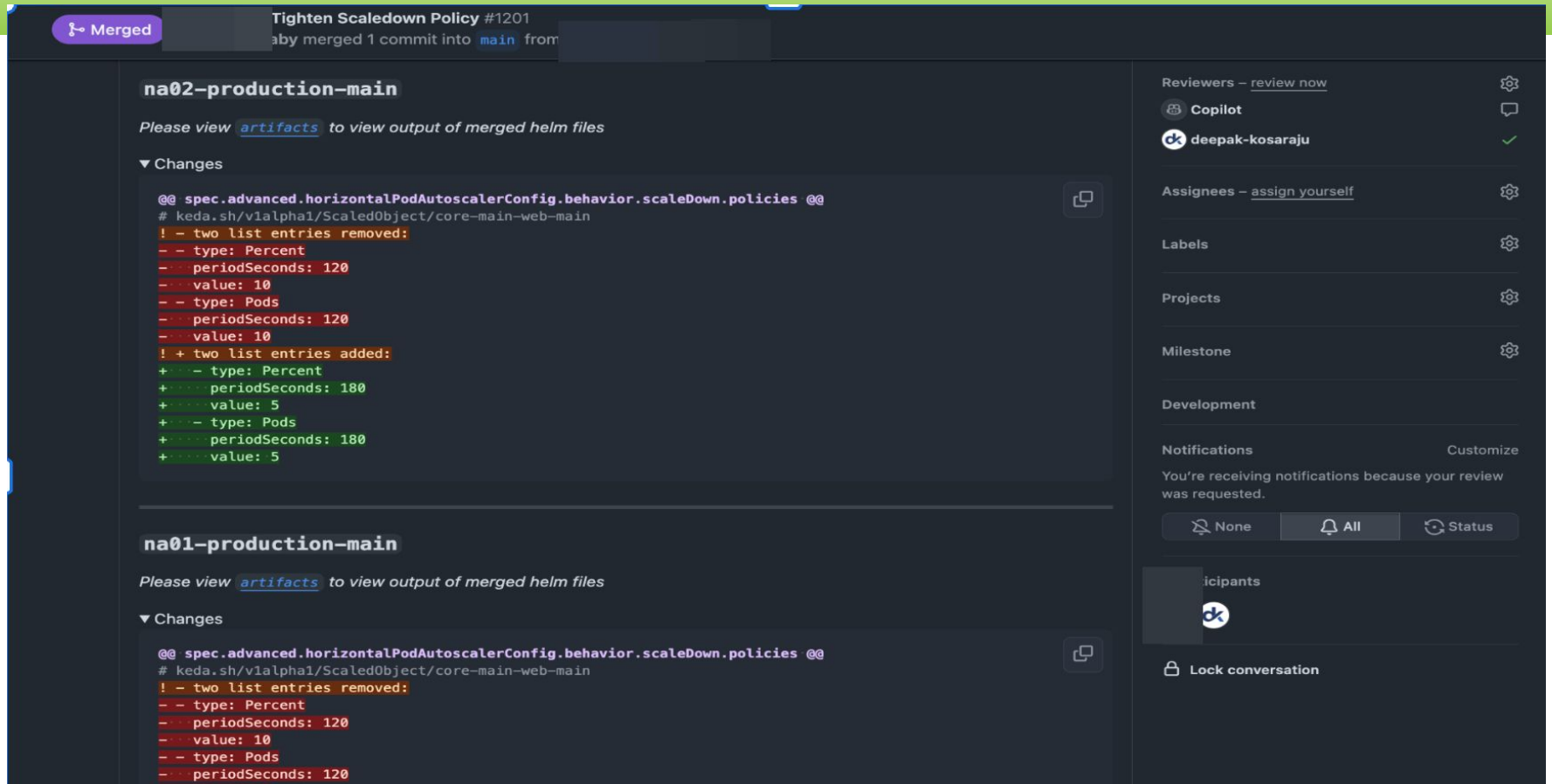
Q2 ‘25
Canary Completion & Karpenter Optimization

Complete full canary rollout implementation.

Experiment with spot instance ratio splits to further reduce operational costs.

Our Design Principles

Our Design Principles: Deploy with Confidence



Our Design Principles - Using ArgoCD

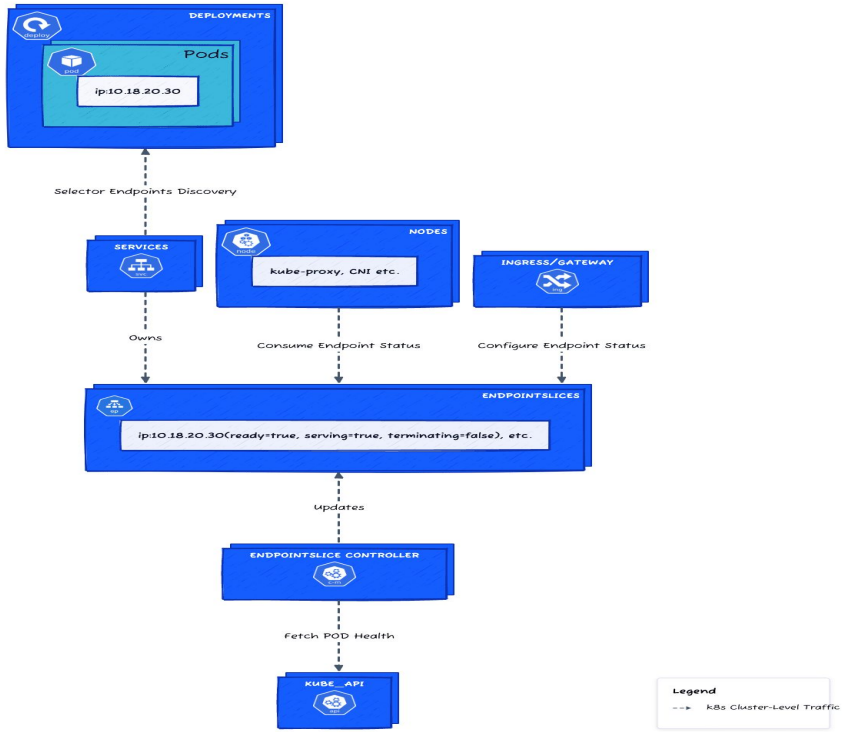
- Bootstrap Controllers and CRDs
- Application Deployment

```yaml
#  ===== Helm Chart Configuration ==========================================================================
sources:
  # Helm Charts Values come from `helm.chart.yaml`.
  # This is templated to allow to override Helm Chart Version in the cluster-specific `helm.chart.yaml` for testing/rollout purposes.
  - repoURL: '{{.chart.repoURL}}'
    chart: '{{.chart.name}}'
    targetRevision: '{{.chart.version}}'
    helm:
      releaseName: 'envoy-gateway'
      # Helm Values loaded according to the directory structure
      valueFiles:
        - $helmValues/cluster-addons/default/envoy-gateway/helm.values.yaml
        - $helmValues/cluster-addons/{{ .values.pirShort }}/default/envoy-gateway/helm.values.yaml
        - $helmValues/cluster-addons/{{ .values.pirShort }}/{{ .values.environment }}/default/envoy-gateway/helm.values.yaml
        - $helmValues/cluster-addons/{{ .values.pirShort }}/{{ .values.environment }}/{{ .values.system }}/default/envoy-gateway/helm.
      values.yaml
        - $helmValues/cluster-addons/{{ .values.pirShort }}/{{ .values.environment }}/{{ .values.system }}/{{ .values.deployment }}/
      default/envoy-gateway/helm.values.yaml
        - $helmValues/cluster-addons/{{ .values.pirShort }}/{{ .values.environment }}/{{ .values.system }}/{{ .values.deployment }}/
      {{ .values.id }}/envoy-gateway/helm.values.yaml
      ignoreMissingValueFiles: true      Youssef Rabie, 4 months ago via  PR #667 • Add envoy gateway as a cluster addon across all…

    # This Source used to load the default values.yaml and the cluster-specific values.yaml for the chart.
  - repoURL: '{{index .metadata.annotations "argocd.procore.com/cluster-addons-repo-url"}}'
    targetRevision: '{{index .metadata.annotations "argocd.procore.com/cluster-addons-repo-revision"}}'
    ref: helmValues

#  ===== Application Sync Configuration ====================================================================
syncPolicy:
  automated:
    prune: true
    selfHeal: true
```

Platform Operators & Controllers

Storage	Workload Management	Observability	Network	Ingress	Security/ Governance	Cluster Hygiene
aws-ebs-csi	Karpenter and it's CRDs Keda Argo Rollouts	Datadog Prometheus Otel Collector	Cilium CoreDNS(Cluster) node local DNS(CoreDNS) ExternalDNS	Envoy Gateway AWS LBC	Kyverno Trivy RBAC Manager External Secrets Cert Manager	spegel deescheduler draino node-problem-detector reloader overprovisioner

Kubernetes Objects/Resources & Components



Gotcha#1 Deploy Rollout

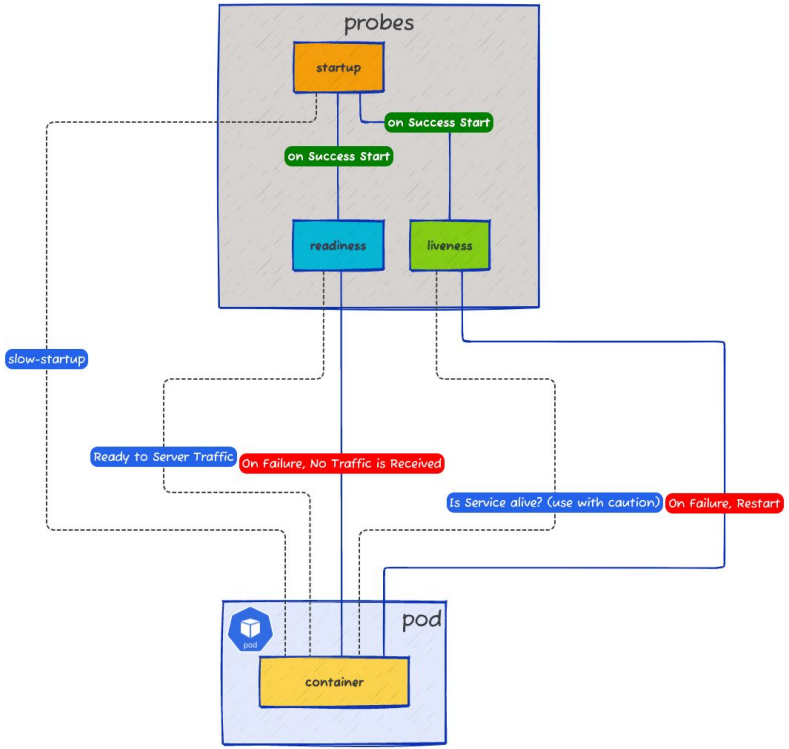
Following primitives are crucial for balancing **deployment time**, **cost**, and **application availability**.

maxSurge, maxUnavailable - k8s Deployment/Rollout strategy attributes
batch-idle-duration and batch-max-duration - Karpenter configuration attributes

For scale Monolith App experiment with above settings to get desired state of your developers expectations.

terminationGracePeriodSeconds == maxRequestTimeout(in your App)

#Gotcha# 2 Health Probe



#Gotcha# 2 Health Probe

Startup	Readiness	Liveness
skip initialDelaySeconds	**Should run more frequent than Liveness**	**It's shouldn't be same as Readiness**
periodSeconds being 1 Sec	periodSeconds:2-5sec	**periodSeconds >= entire failureThreshold of Readiness**
failureThreshold is 2x of your avg app boot time	You can mostly rely on default settings for others	You can mostly rely on default settings for others

Note: All Probe defaults are disruptive, so pay attention to these settings

Great articles, worth reading.

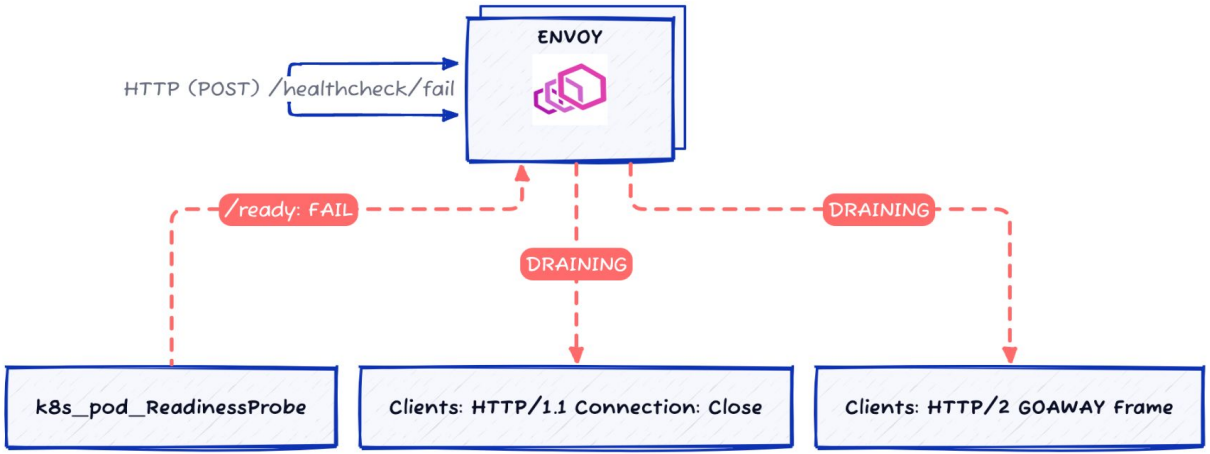
https://blog.eightnoteight.dev/2023/02/26/health-checking/ - 2023

https://careersatdoordash.com/blog/how-to-handle-kubernetes-health-checks/ - 2022

https://srcco.de/posts/kubernetes-liveness-probes-are-dangerous.html - 2019

Gotcha# 3 Envoy Proxy

Envoy offers an admin endpoint /healthcheck/fail to help with draining inflight requests.
DO NOT send POST to /drain_listeners?graceful as it will have other effects.

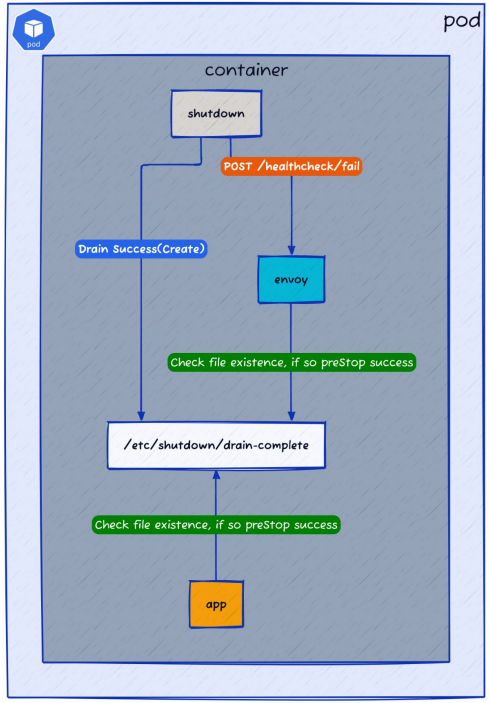


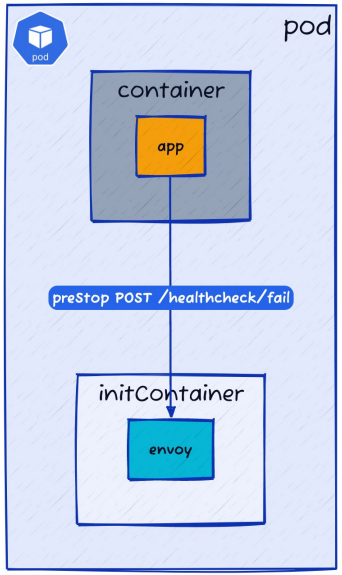
Gotcha#3 Container Lifecycle Hooks

preStop	postStart
• Hold's SIGTERM to containers.	• We use this to send POST request to envoy proxy /healthcheck/ok
• Helps for any cleanup tasks, in our case ○ We drain any active connections by sending a POST request to envoy proxy /healthchecks/fail which signal client to close connections.	• We use this to send POST request to envoy proxy /healthcheck/ok endpoint ensuring envoy readiness checks pass and app can server traffic again.

We use envoy proxy as sidecar and if app container did restart for reasons where liveness check failed etc.. then we have to ensure during postStart hook we have to undo any actions that were triggered against envoy proxy during preStop

Gotcha#3 Total 3 Iterations





Gotcha# 3 preStop Iteration 1

emptyDir + Shutdown Sidecar


```
    envoy:
      lifecycle: *wait-for-shutdown-lifecycle
      # Empty Directories shared between the containers of the same pod.
      # -- Key must match a volume name defined in the `sharedVolumes` section.
      mountedContainerSharedDir:
        shutdown-signal-dir:
          - mountPath: /etc/shutdown/
  sidecars:
    shutdown-orchestrator:
      image:
        repository: "busybox"
        tag: "1.35"
      # Sleep forever but react to SIGTERM.
      command: [ /bin/sh, -c, "trap : TERM INT; sleep infinity & wait"]
      resources:
        requests:
          cpu: "3m"
          memory: "10Mi"
        limits:
          memory: "10Mi"
      mountedContainerSharedDir:
        shutdown-signal-dir:
          - mountPath: /etc/shutdown/
      lifecycle:
        # This is the `primary` preStop hook. # It's main purpose is to:
        # 1. Trigger Envoy to drain all active connections and signal ALB to close connections.
        #   1.1 Envoy will stop accepting new ones after --drain-time-s (60) seconds.
        # 2. Wait for few (30) seconds to wait until Kubernetes stop sending traffic to the pod. (for AWS LBC to update LB Target Groups, etc)
        # 3. Wait for all active connections to drain.
        # 4. Signal other container's preStop hooks to continue and to cleanly shutdown.
        # 5. ... Kubernetes will then send SIGTERM to the container (and all other containers).
        # Note: - We wait a total of `30` seconds at All times to ensure Kubernetes will not send any new requests to this pod.
        #         The `30` second sleep must be <= `--drain-time-s` value in the envoy container.
        # ----
        # ⚠️ Do not change these values unless you understand the shutdown sequence and the implications of changing it.
        preStop:
          exec:
            command:
              - sh
              - -c
              - >
                echo "[Kubernetes PreStop] Starting Drain Sequence." > /proc/1/fd/1 2>&1;
                echo "[Kubernetes PreStop] Signaling Envoy to Stop Accepting Connections." > /proc/1/fd/1 2>&1;
                wget -qO - --post-data '' "http://envoy:9901/drain_listeners?graceful"; sleep 30;

                echo "[Kubernetes PreStop] Waiting for Active Connections to Drain. (if any)" > /proc/1/fd/1 2>&1;
                while [[ $(wget -q -O- envoy:9901/stats | grep http.ingress_https.downstream_cx_active | awk '{print $2}') != 0 ]];
                do sleep 1; done;

                echo "[Kubernetes PreStop] All Connections Drained... Signaling Sidecars to Shutdown" > /proc/1/fd/1 2>&1;
                touch /etc/shutdown/drain-complete;
```

Gotcha# 3 preStop Iteration 1

emptyDir + Shutdown Sidecar


```yaml
containers:
  app:
    # --- Lifecycle Hooks ----------------------------------------------------------------------------------------------
    lifecycle: &wait-for-shutdown-lifecycle
      preStop:
        exec:
          command:
            - bash
            - -c
            # Sleep 21 since the `shutdown-orchestrator` container will wait for 20 seconds for the connections to drain anyway. (Avoid logging unnecessarily)
            - >
              sleep 21;
              if [[ ! -f /etc/shutdown/drain-complete ]]; then echo "[Kubernetes PreStop] Waiting for Drain Completion" > /proc/1/fd/1 2>&1; fi;
              while [[ ! -f /etc/shutdown/drain-complete ]]; do sleep 1; done;
              echo "[Kubernetes PreStop] Drain Completed" > /proc/1/fd/1 2>&1;
    # Empty Directories shared between the containers of the same pod.
    # -- Key must match a volume name defined in the `sharedVolumes` section.
    mountedContainerSharedDir:
      shutdown-signal-dir:
        - mountPath: /etc/shutdown/
  envoy:
    lifecycle: *wait-for-shutdown-lifecycle
    # Empty Directories shared between the containers of the same pod.
    # -- Key must match a volume name defined in the `sharedVolumes` section.
    mountedContainerSharedDir:
      shutdown-signal-dir:
        - mountPath: /etc/shutdown/
```

https://github.com/deepak-kosaraju/kubecon25-zero-downtime/tree/main/platform-services/web-app/lifecycle-hooks/before-native-sidecar

Gotcha# 3: preStop Iteration 2

Native Sidecars (k8s v1.28) + Ingress Controller

- Switched to native sidecars, removed `emptyDir` dependency and custom shutdown sidecar to orchestration of connection draining and TERM sequence.
- Switched to using /healthcheck/fail instead of /drain_listeners.

Gotcha# 3: preStop Iteration 3

L7 Gateway + Aggressive Deletion

- With `Envoy Gateway` we have observed that EndpointDiscoveryService(EDS) is removing POD's from cluster endpoint's 10x faster than AWS-LBC-Controller reconciliation.

Demo

- Lifecycle Hooks for graceful termination of POD
  - Easy(sleep X) vs
  - Most reliable and accurate preStop solutions for connection draining and graceful termination

https://github.com/deepak-kosaraju/kubecon25-zero-downtime

Gotcha# 5 Horizontal Pod Autoscaler with KEDA

KubeCon CloudNativeCon North America 2025

HPA vs. KEDA Reaction Time

- KEDA's pollingInterval is flexible, but EKS HPA's horizontal-pod-autoscaler-sync-period (e.g., 30s) is what matters the post when scaling from 1 to N.
- horizontal-pod-autoscaler-sync-period is your ***true minimum reaction time*** for scaling actions.

Built-in Stability

- HPA has a +/- 10% tolerance on the desired replica count. This prevents "flapping" (rapid, unneeded scaling) from minor metric noise.

  (Note: This is a configurable field in Kubernetes 1.33+ (Alpha)).

Gotcha# 5 Horizontal Pod Autoscaler with KEDA

Scale-Up Strategy (For "Slow Boot" Apps)

- Scale *proactively*. Set your HPA metric trigger at 60% or 70% of your target metric tolerance.
- This gives slow-starting pods time to become ready *before* you hit maximum load.

Gotcha# 5 Horizontal Pod Autoscaler with KEDA

Scale-Down Strategy (For "Slow Shutdown" Apps)

- Define custom scaleDown policies (using HPA behavior).
- This lets you control how many pods you can safely lose at once.
- **Action:** Review and adjust these policies regularly based on your traffic patterns.

The Safety Net (KEDA fallback)

- Risk: If your metrics provider (e.g., Datadog) fails, HPA defaults to the *current* replica count.
- Solution: Set fallback replicas. We recommend 80% of maxReplicas to prevent an outage.

Gotcha# 5 Horizontal Pod Autoscaler with KEDA

The Safety Net (KEDA fallback)

- Risk: If your metrics provider (e.g., Datadog) fails, HPA defaults to the *current* replica count.
- Solution: Set fallback replicas. We recommend 80% of maxReplicas to prevent an outage.

Gotcha# 6 ArgoRollout

- We use Basic Canary and not Traffic Switching due to known issues around Unsafe Traffic Switching during rollback and mid-rollout-update
- maxUnavailable budget isn't violated based on your setWeight **%**
- maxSurge to tradeoff between deployment time vs cost.
- Any HPA Scale would add to canary replicas unless we are in pause phase of deployment

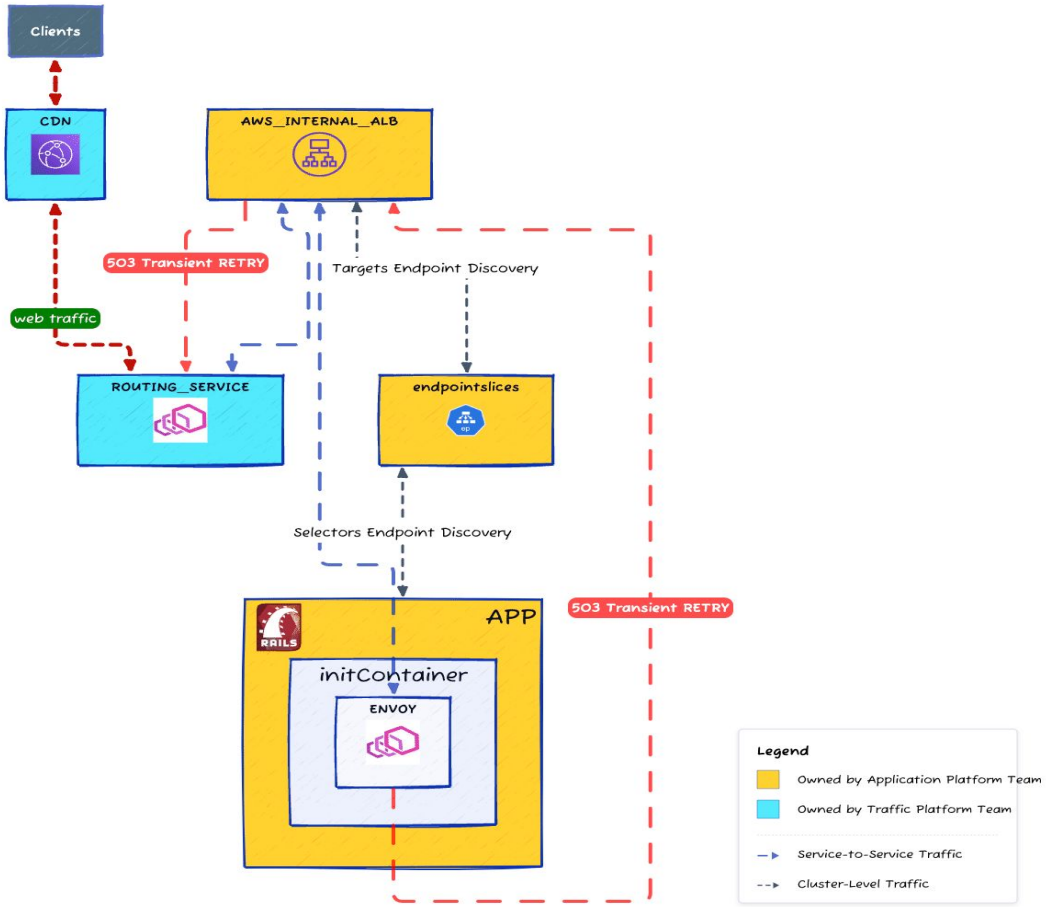
Gotcha# 7 Can We Eliminate 503 Errors Completely?

Yes and it's a Tradeoff between Level of Efforts to Optimize the App vs Cost(which involved Over-Provisioning the Cluster)

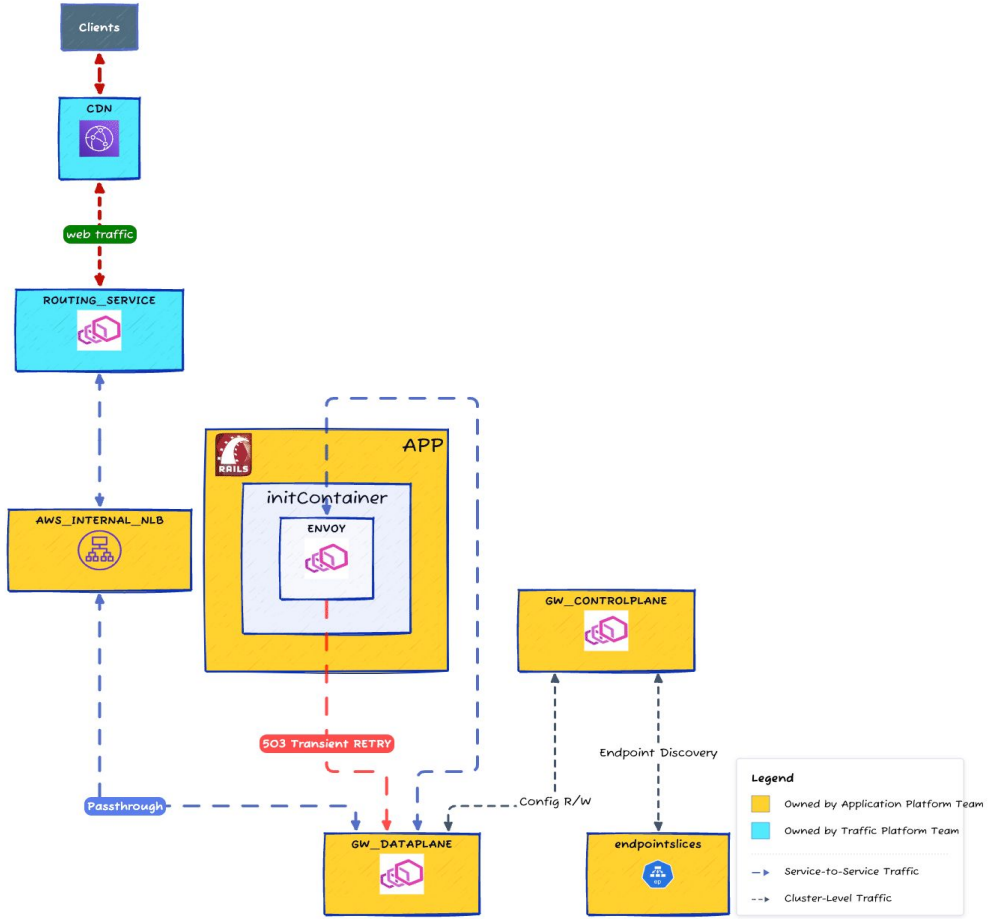
Four Layers influences it,

1. Container Image Size & Application Boot Time.
2. Traffic Distribution/Split using Layer 7 Routing
3. Cluster Autoscaling
4. Application Autoscaling which depends on 1 and 2

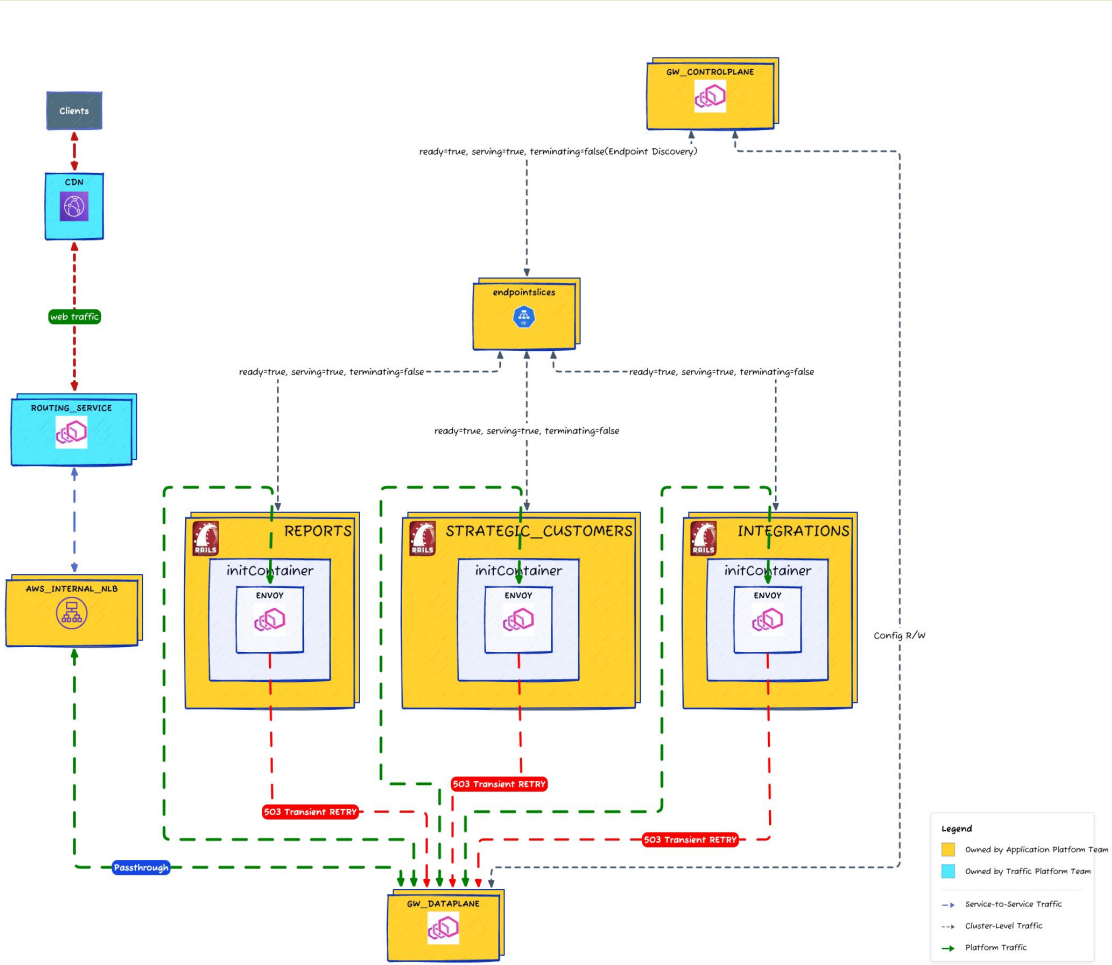
Old Topology



Current Topology



Future Topology



Demo

- Do not serve 503 to customers but handle them through transient retry with-in the Platform

https://github.com/deepak-kosaraju/kubecon25-zero-downtime/tree/main/platform-services/dataplane

Open Source Software(OSS) Contributions

Youssef Rabie
@y-rabie

Thanks to Youssef and Hesham for their contribution to Open Source Software(OSS). Between these two engineers from our team there were **17 contributions with 15 merged across 4 OSS projects in last 1 year.**

- KEDA
- Argo Rollouts
- AWS-LBC
- k9s

Hesham Sherif
@heshamelsherif97

Key Takeaways & Testing Strategy

1 Executive Leadership Buy-in

Ensure **strong support and commitment from leadership** for deployment strategy initiatives.

2 Observability First

Define tagging and observability strategy before implementation to ensure proper monitoring and debugging capabilities. **Measure twice, cut once.**

3 Test Success AND Failures

Experiment with both success and failure scenarios in **playground/developer environments before production deployment.**

4 Optimize for Quick Iterations

Implement **mechanical diff reviews for cluster configurations** and deployment manifests to streamline validation processes.

Open to Questions?

KubeCon CloudNativeCon North America 2025

Thank You to all colleagues at Procore and you all for making time to attend making this possible.

Feedback Appreciated