DevOpsCon

by devmio

Revolutionizing Platform Engineering with GitOps and ArgoCD

Maryam Tavakkoli
Senior Cloud Engineer @ Relex Solutions
December 2024

Who Am I?

- Maryam Tavakkoli
- Senior Cloud Engineer @ RELEX Solutions
- CNCF Ambassador
- Microsoft MVP
- Kubernetes & CNCF Meetup Co-organizer
- LinkedIn: maryam-tavakkoli
- Medium: @maryam.tavakoli.3

Agenda

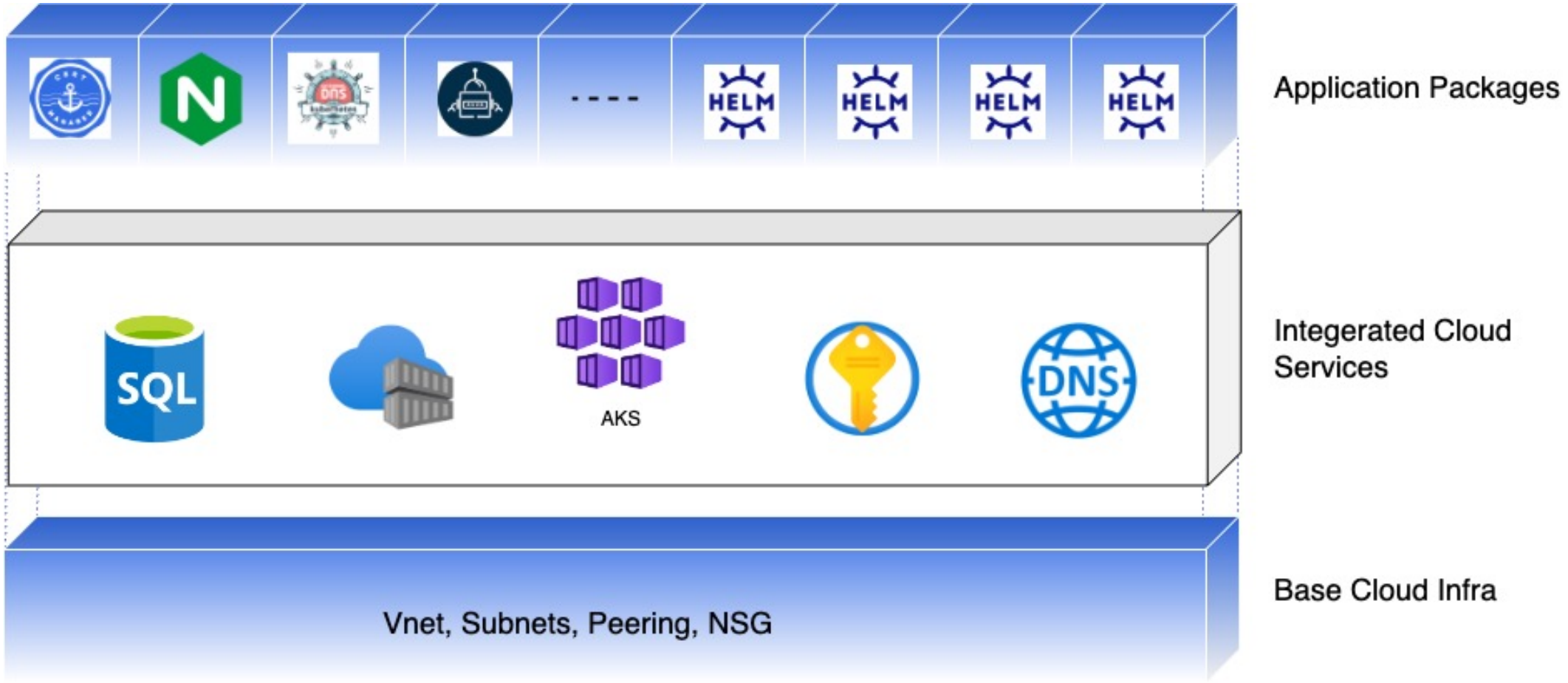
- Introduction to Relex's Internal Kubernetes Platform
- Platform v1.0: Challenges and Limitations
- Revolutionizing with GitOps
- Designing GitOps for Large-Scale Platforms: Key Challenges
- Summary and Key Takeaways

Introduction to
Relex's Internal Kubernetes Platform

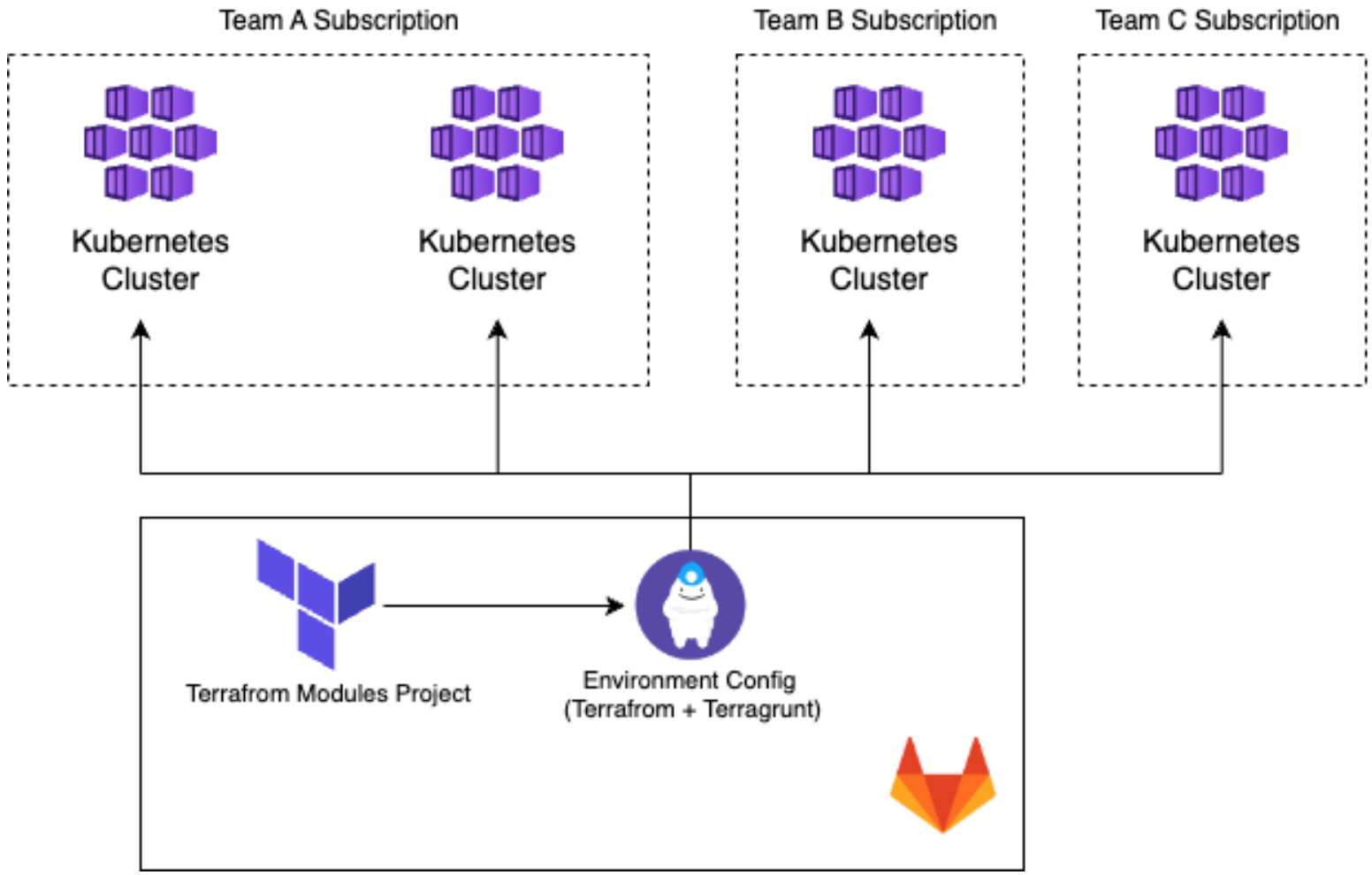
The Architecture

- SaaS company
- Around 13 different Dev teams users of platform
- Azure Kubernetes service (AKS) at core
- Azure services and CNCF/commercial Projects
- Metrics, logging, secret management and ingress traffic management
- Multiple optional enhancements (add-ons)

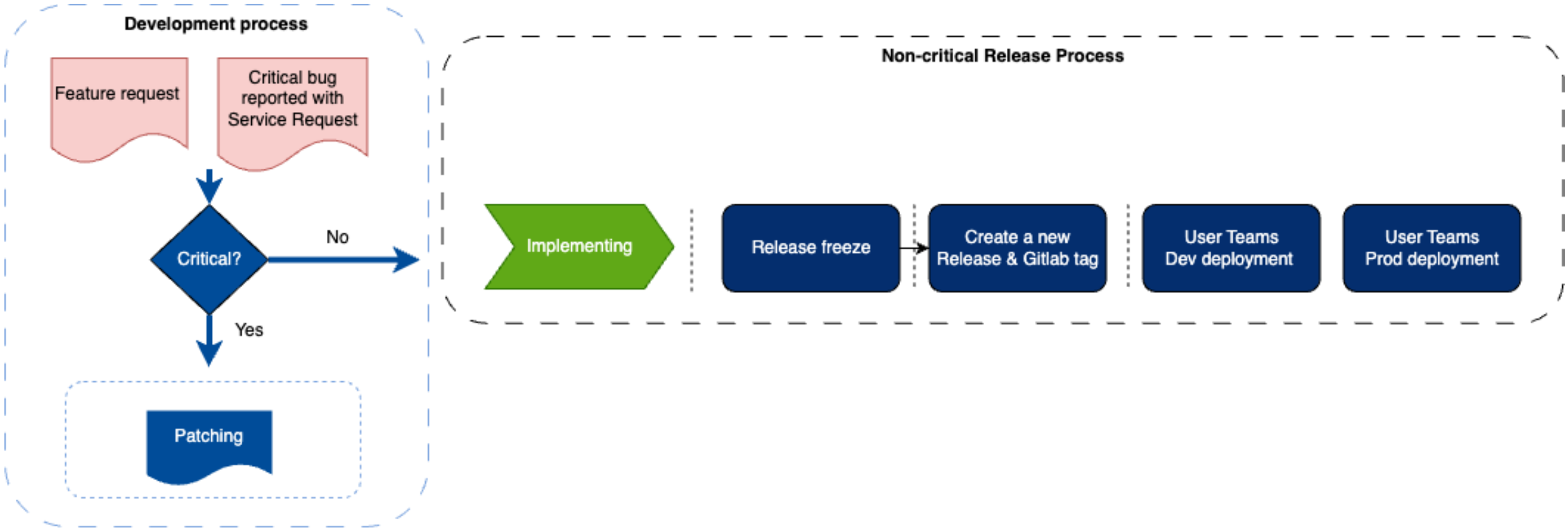
Internal Kubernetes Platform Architecture



Deploying Instances of the IDP



Release Model



Platform v1.0:
Challenges and Limitations

Platform v1.0: Challenges and Limitations

- Excessive Releases:
  - Users express concerns about the high frequency of releases.
- Core Stability vs. Change:
  - Core services and infrastructure require minimal updates.
  - Applications and cluster add-ons demand frequent upgrades, primarily for security compliance.
- Helm App Configurations:
  - Introducing new user configurations for Helm applications often necessitates additional patches for existing releases.

Revolutionizing with GitOps

Introduction to GitOps

The definition of GitOps, as outlined by gitops.tech:

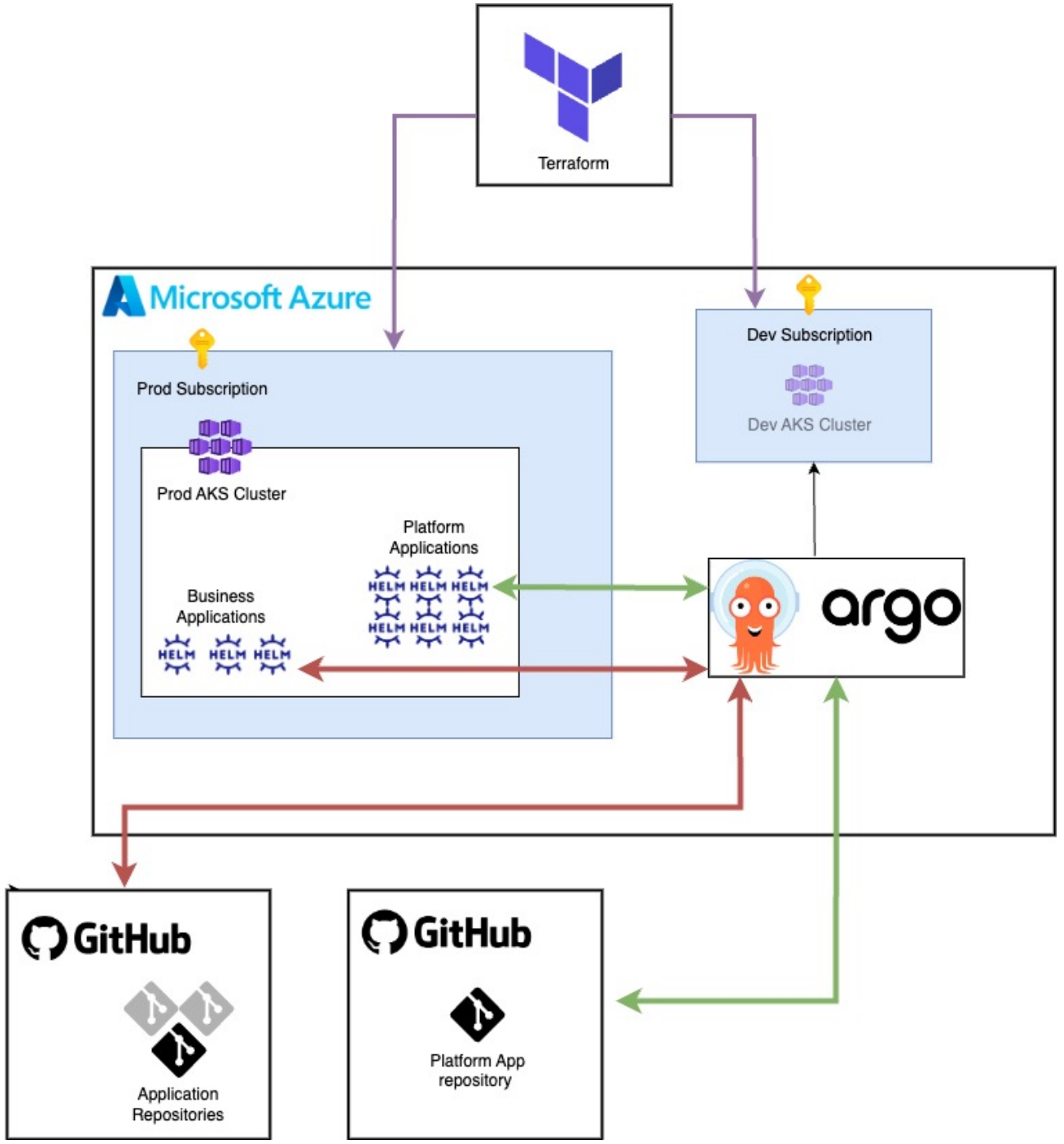
The core idea of GitOps is having a Git repository that always contains declarative descriptions of the infrastructure currently desired in the production environment and an automated process to make the production environment match the described state in the repository. If you want to deploy a new application or update an existing one, you only need to update the repository - the automated process handles everything else. It's like having cruise control for managing your applications in production.

ArgoCD

ArgoCD is a declarative, GitOps continuous delivery tool for Kubernetes. It is designed to facilitate the deployment and management of applications in a Kubernetes cluster using Git as the source of truth for the desired application state.

ArgoCD as a GitOps tool

Architecture in Platform v2.0



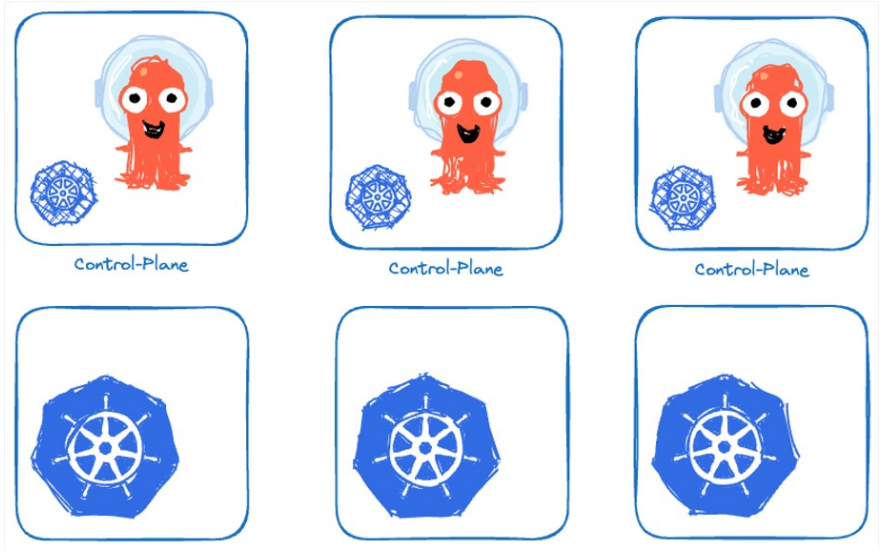
Designing GitOps for Large-Scale Platforms: Key Challenges

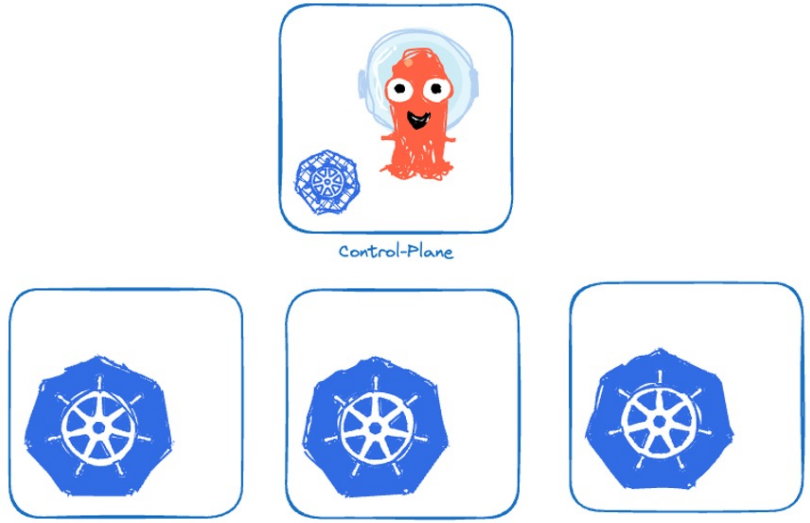
Designing GitOps for Large-Scale Platforms

- Control Plane strategy
- Self-management of ArgoCD
- Child cluster registration
- Foundational design principles

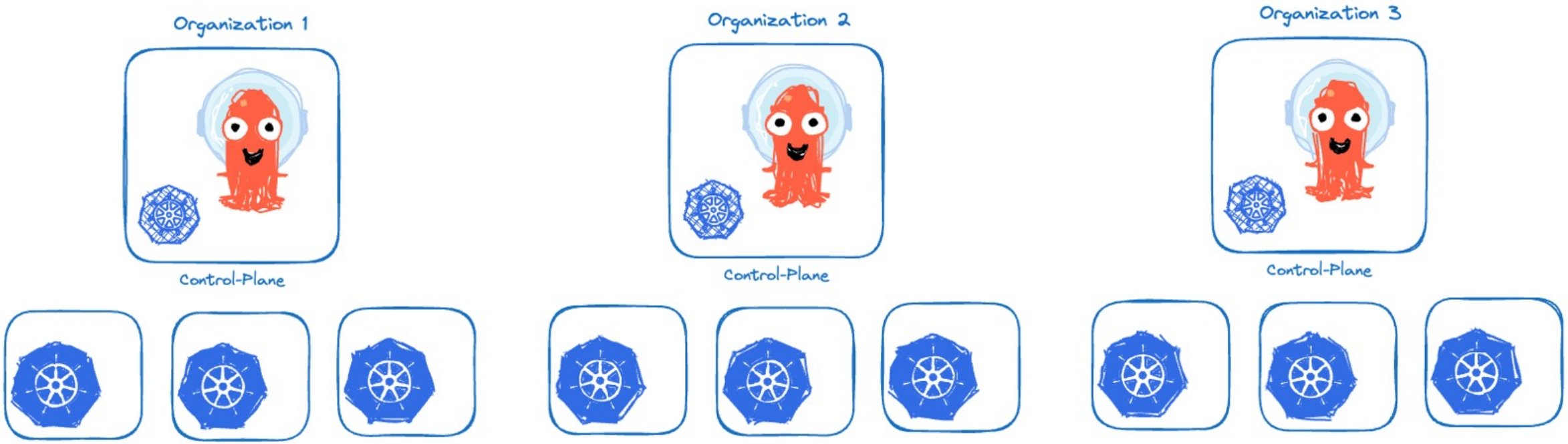
Control Plane Strategy

- Implementing control planes for clusters across the entire company.
- Defining ownership and management of ArgoCD control planes.





Control Plane Strategy



Self-Management of ArgoCD

- You can install ArgoCD with a helm chart and let ArgoCD manage it after the first-time installation using App-of-Apps pattern.
- App-of-apps pattern is to define a parent application that manages multiple child applications. You can use this to manage installation of ArgoCD.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argo-cd
  namespace: argocd
spec:
  project: argocd
  source:
    repoURL: https://xxx.git
    targetRevision: main
    path: argocd-control-planes/platform-dev-argo-cp
  destination:
    server: https://kubernetes.default.svc
    namespace: argocd
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

ArgoCD Application for managing ArgoCD

```yaml
helmCharts:
- name: argo-cd
  repo: https://argoproj.github.io/argo-helm
  version: 6.2.0
  releaseName: argocd
  namespace: argocd
  valuesFile: base/values.yaml

resources:
- base/application-argocd.yaml
- base/git-generator-helm-charts.yaml
- base/git-generator-charts-resources.yaml
- base/namespaces.yaml
- base/projects.yaml
```

Kustomization.yaml including ArgoCD Helm chart

Child Cluster Registration

External Secret Operator (ESO)

An open-source tool designed to synchronize Kubernetes Secrets with external secret management systems like Azure Key Vault (AKV).

```yaml
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: argocd-control-plane-cluster-secret
  namespace: argocd
spec:
  refreshInterval: 1h
  secretStoreRef:
    kind: ClusterSecretStore
    name: azure-cluster-secret-store
  target:
    name: argocd-control-plane-cluster-secret
    template:
      engineVersion: v2
      metadata:
        labels:
          argocd.argoproj.io/secret-type: cluster
```

```
  template:
    engineVersion: v2
    metadata:
      labels:
        argocd.argoproj.io/secret-type: cluster    # This label is required for cluster registeration
    templateFrom:
    - target: Annotations  # To pass helm chart values as annotations
      literal: |-
        {{ $parsed := fromJson .secret }}
        {{ range $key, $value := $parsed }}
        {{ $key }}: {{ $value }}
        {{ end }}
    - target: Data # To pass cluster name and host as data in the k8s secret
      literal: |-
        {{ $parsed := fromJson .clusterdata }}
        {{ range $key, $value := $parsed }}
        {{ $key }}: {{ $value }}
        {{ end }}
    data:  # To pass cluster config as data in the k8s secret
      config: "{{ .app }}"
data:
  - secretKey: secret
    remoteRef:
      key:  charts-info
  - secretKey: clusterdata
    remoteRef:
      key:  cluster-info
  - secretKey: app
    remoteRef:
      key:  cluster-config
```

Foundational design principles

- Leverage Upstream Helm Charts
  - Avoid maintaining custom charts by using upstream charts as the baseline.
- Platform-Level Value Overwrites
  - Apply standardized value overwrites to upstream charts to meet platform-wide requirements for all users.
- Environment-Specific User Customizations
  - Allow users to overwrite chart values for their specific environments, ensuring flexibility.

Foundational design principles

- User-Specific Folder Structures
  - Provide each user with a dedicated space (folder) in the shared repository, enabling:
    - Codeowner-based access control for pull request reviews.
    - Clear organization and accountability.
- Focus on Maintainability
  - Minimize repetition and ensure scalable, sustainable configurations.

Foundational design principles

- Terraform Integration Challenges
  - Reduce the reliance on hard-coded values by enabling integration between Terraform and ArgoCD.
  - Automate the passing of relevant Azure resources created by Terraform (e.g., storage accounts, DNS zones, Workload Identities) directly into ArgoCD for deployment.

Repository Structure

Layered Configuration Management

```yaml
apiVersion: v2
name: cert-manager
type: application
version: 1.0.0
dependencies:
  - name: cert-manager
    version: v1.12.1
    repository: https://charts.jetstack.io
```

Umbrella Chart (root of the repository)

```yaml
cert-manager:
  podAnnotations:
    ad.datadoghq.com/controller.logs: |
      [
        {
          "source": "cert-manager",
          "service": "controller"
        }
      ]
    ad.datadoghq.com/tags: |2
      {
        "sourcecategory": "development"
      }
```

Overwrite chart values in user config folder

ApplicationSet

Generate ArgoCD applications based on

- Available Child Clusters
- Directory Path for Helm Charts

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: addons
  namespace: argocd
spec:
  goTemplate: true
  generators:
  - matrix:
      generators:
        - clusters:
            selector:
              matchLabels:
                argocd.argoproj.io/secret-type: cluster
        - git:
            repoURL: https://xxx.git
            revision: main
            directories:
            - path: helm-charts/*
```

ApplicationSet

```yaml
template:
  metadata:
    name: '{{.path.basename}}-{{.name}}'
  spec:
    project: '{{.metadata.annotations.team}}'
    source:
      repoURL: https://xxx.git
      targetRevision: main
      path: '{{.path.path}}'
      helm:
        valueFiles:
          - values.yaml
          - "../../argocd-control-planes/platform-dev-argo-cp/environments/{{.name}}/{{.path.basename}}/values.yaml"
        values: |-
```

Terraform Integration Challenges

- Resources like workload identities for Helm charts are created in Terraform and need to be passed to ArgoCD.
- ArgoCD ApplicationSet can utilize data from ***annotations*** within the ***cluster registration secret*** to manage these dependencies.
- Pass the required data into the Kubernetes secret used for cluster registration.
- Access these values in ApplicationSet by referencing *.metadata.annotations.xxx.*
- This innovative approach is credited to gitops-bridge-dev.

Terraform Integration Challenges

```
helm:
  valueFiles:
    - values.yaml
    - "../../argocd-control-planes/platform-dev-argo-cp/environments/{{.name}}/{{.path.basename}}/values.yaml"
  values: |-
    {{- if eq .path.basename "external-dns" }}
      external-dns:
        serviceAccount:
          annotations:
            azure.workload.identity/client-id: {{.metadata.annotations.externaldnswi}}
        azure:
          resourceGroup: {{.metadata.annotations.externaldnsrg}}
          subscriptionId: {{.metadata.annotations.externaldnssubscription}}
    {{- end }}
    {{- if eq .path.basename "cert-manager" }}
      cert-manager:
        serviceAccount:
          annotations:
            azure.workload.identity/client-id: {{.metadata.annotations.certmanagerwi}}
    {{- end }}
```

```
  template:
    engineVersion: v2
    metadata:
      labels:
        argocd.argoproj.io/secret-type: cluster    # This label is required for cluster registeration
    templateFrom:
    - target: Annotations  # To pass helm chart values as annotations
      literal: |-
        {{ $parsed := fromJson .secret }}
        {{ range $key, $value := $parsed }}
        {{ $key }}: {{ $value }}
        {{ end }}
    - target: Data # To pass cluster name and host as data in the k8s secret
      literal: |-
        {{ $parsed := fromJson .clusterdata }}
        {{ range $key, $value := $parsed }}
        {{ $key }}: {{ $value }}
        {{ end }}
    data:  # To pass cluster config as data in the k8s secret
      config: "{{ .app }}"
data:
  - secretKey: secret
    remoteRef:
      key:  charts-info
  - secretKey: clusterdata
    remoteRef:
      key:  cluster-info
  - secretKey: app
    remoteRef:
      key:  cluster-config
```

Summary and Key Takeaways

Summary and Key Takeaways

- GitOps Enables Scalability
  - Automating cluster management and application deployments with GitOps significantly reduces manual effort and operational bottlenecks, allowing for smoother scaling.
- Strategic Design for Long-Term Success
  - A well-thought-out architecture ensures both long-term maintainability and scalability as the platform grows.
- Terraform Integration for efficiency
  - Integration between Terraform and ArgoCD enhances workflow efficiency.

Summary and Key Takeaways

- Striking the Right Balance
  - Balancing user flexibility with platform-level consistency is essential to maintaining control while enabling developers to work efficiently and autonomously.
- Iterative Improvement is Key
  - A continuous improvement mindset—where processes evolve based on feedback and real-world usage—ensures the platform remains agile and responsive to new challenges.

Thank You!

Maryam Tavakkoli

Senior Cloud Engineer @ RELEX Solutions | CNCF Ambassador | Microsoft MVP |...