

Platform Engineering on EKS: A Developer Self- Service GitOps Golden Path

Yuji Matsuoka

Solutions Architect

Amazon Web Services

自己紹介

松岡 雄地

アマゾン ウェブ サービス ジャパン
ソリューションアーキテクト

役割

- ✓ 通信業界のお客様への技術支援

経歴

- ✓ 三重出身 大学で京都に6年
- ✓ NTTDATA
 - ✓ 決済プラットフォーム バックエンドエンジニア
 - ✓ 社内プラットフォーム SRE
 - ✓ CCoE活動、新人育成
 - ✓ 運用改善・自動化・コスト最適化など
 - ✓ SREとして各案件の支援



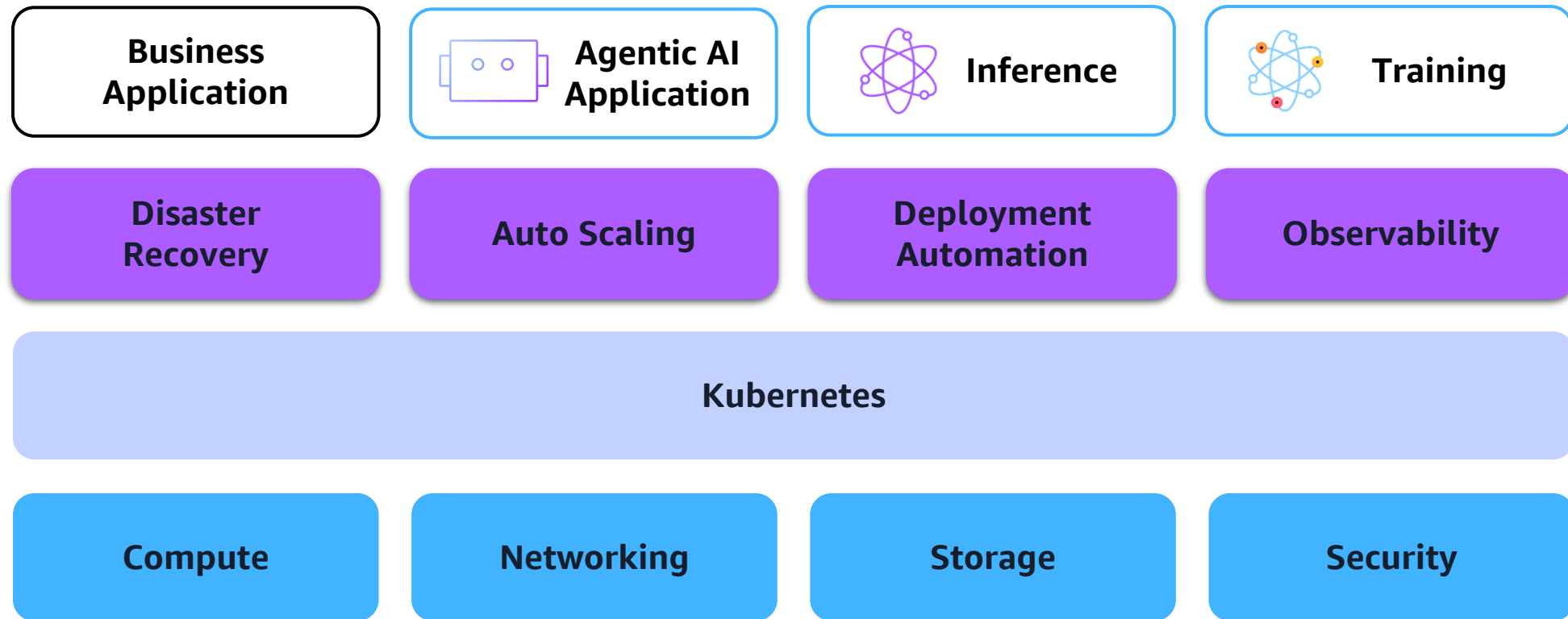
趣味

- ✓ 麻雀、ポーカー、筋トレ

アジェンダ

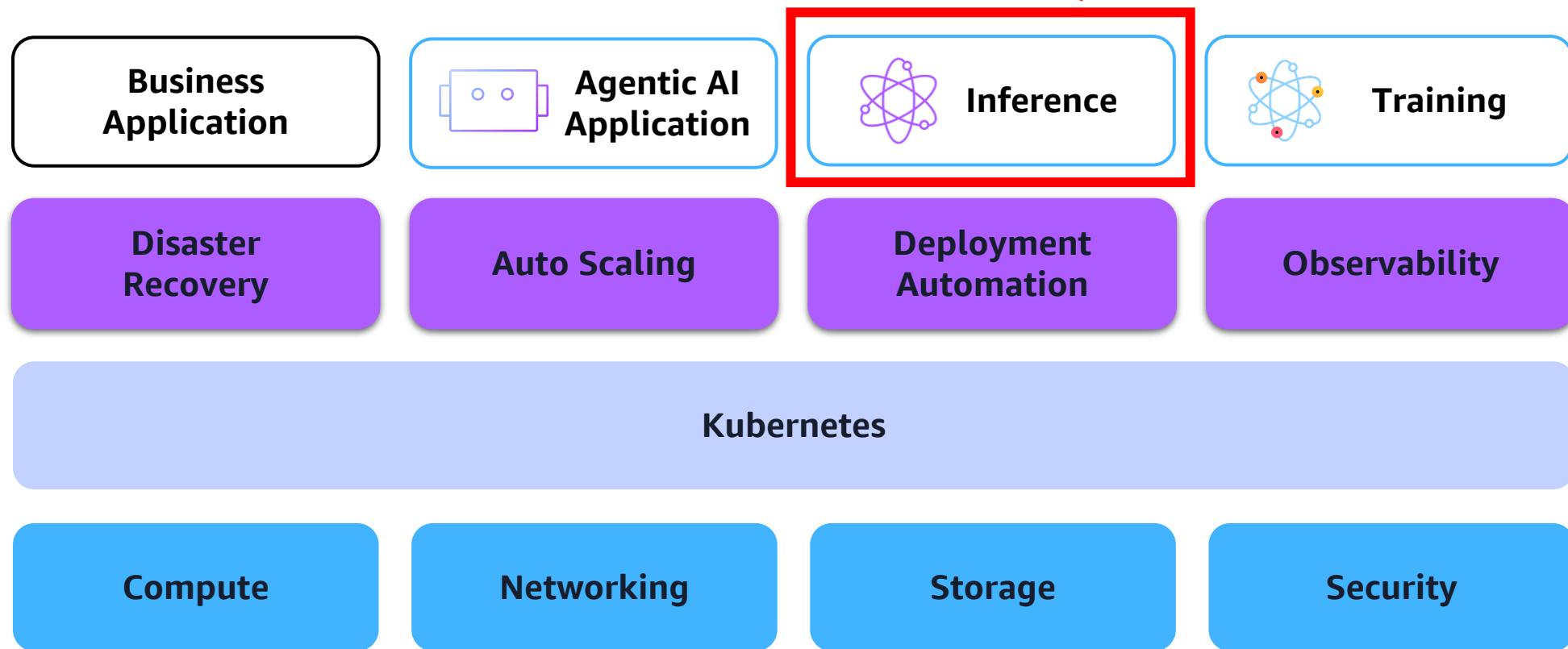
- ✓ Kubernetesを用いた標準化
- ✓ AWSでの推論アーキテクチャ
- ✓ EKS Capabilitiesのご紹介
- ✓ ゴールデンパスのご紹介

Kubernetesを用いた標準化

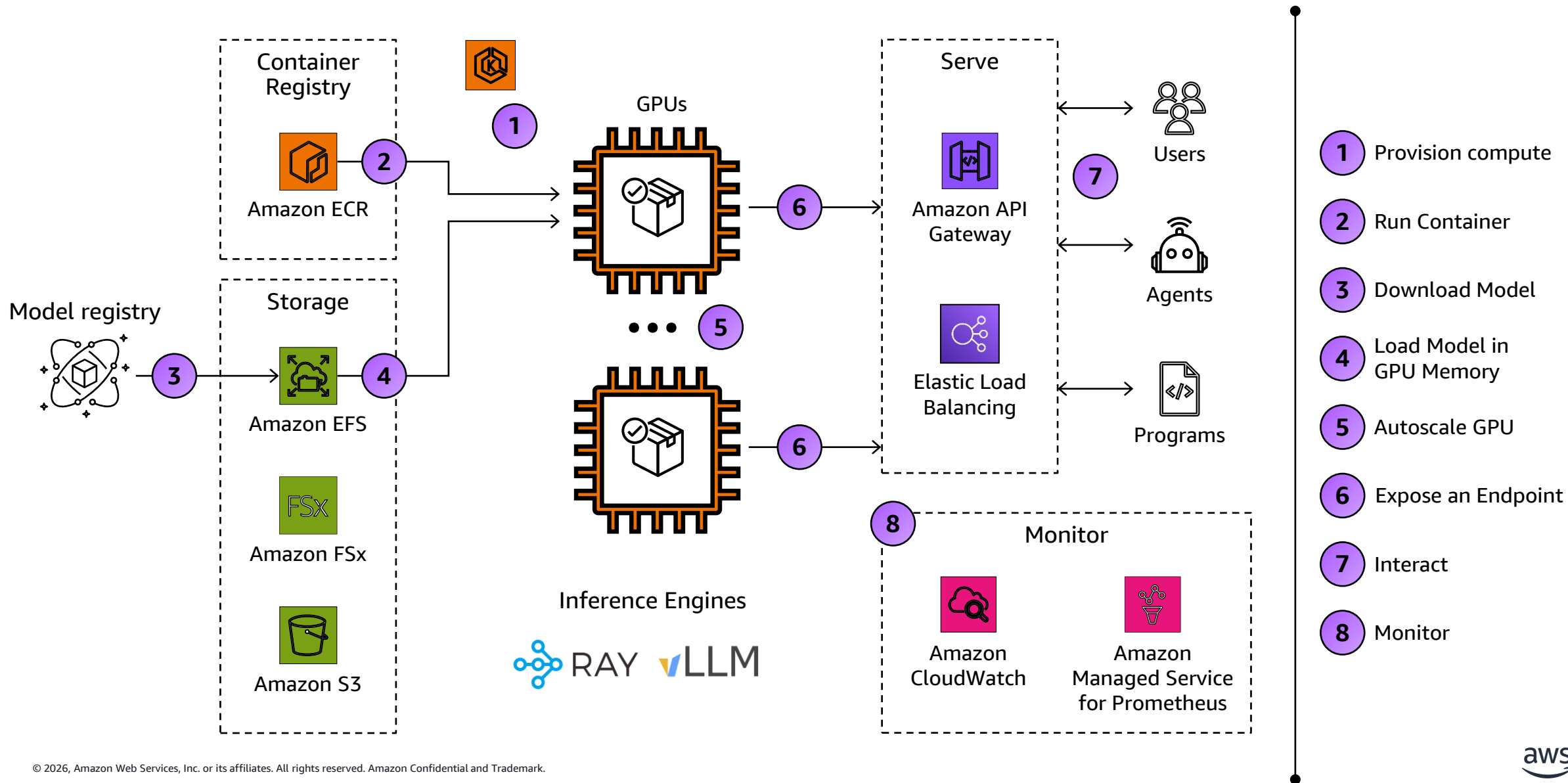


Kubernetesを用いた標準化

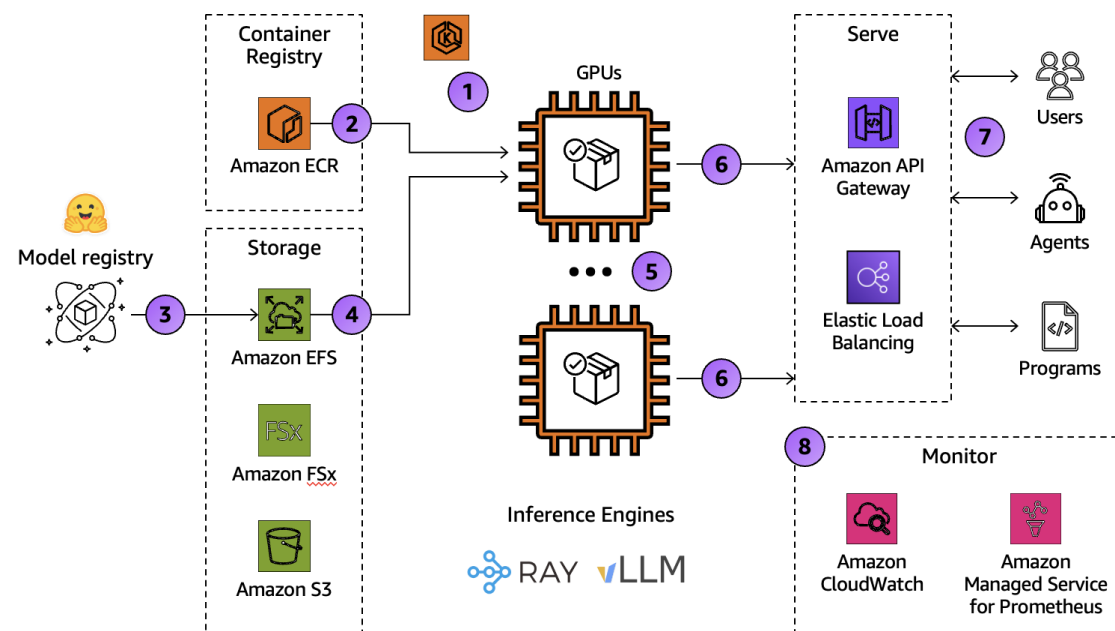
本日例としてお話しするワークロード



AWSでの推論アーキテクチャ



認知負荷の課題



開発者はたった1つのLLMを本番に出すだけで、コンテナ・ストレージ・GPU・推論エンジン・スケール・監視といった構成要素を理解する必要があります。

もちろん理解しているに越したことはないのですが、これはアプリ開発者が本来やるべきこととは関係ない余計な認知負荷を生み出している場合が往々にしてあると思います。

この認知負荷をプラットフォーム側で吸収し、削減する手法を本日は紹介できればと思います。

ソリューション

- 以下の3つは、EKS クラスターに環境構築やスケーリングを効率的に行うためのOSSです。
 - ACK(AWS Controllers for Kubernetes)
 - Argo CD
 - Kro (Kube Resource Orchestrator)



AWS Controllers for Kubernetes
(ACK)

AWSリソース (IAM・S3等) を
Kubernetesマニフェストで宣言的に管理



ArgoCD

GitOpsのツール
Gitの状態をそのままクラスタに反映



Kube Resource Orchestrator
(Kro)

複数リソースを束ねて1つの独自APIに抽象化

Amazon EKS Capabilities

- EKS クラスターに環境構築やスケーリングを効率的に行うための機能をフルマネージドな形で追加可能に
- 現時点で提供されている機能（Capability）は以下の3つ
 - AWS Controllers for Kubernetes (ACK)
 - Argo CD
 - kro (Kube Resource Orchestrator)
- EKS Capabilities を利用することで、これらのインストールやアップグレードといった管理作業が不要に
- ワーカーノード上のCPU・メモリを消費せず、クラスターリソースを最大限活用可能に



AWS Controllers for Kubernetes
(ACK)

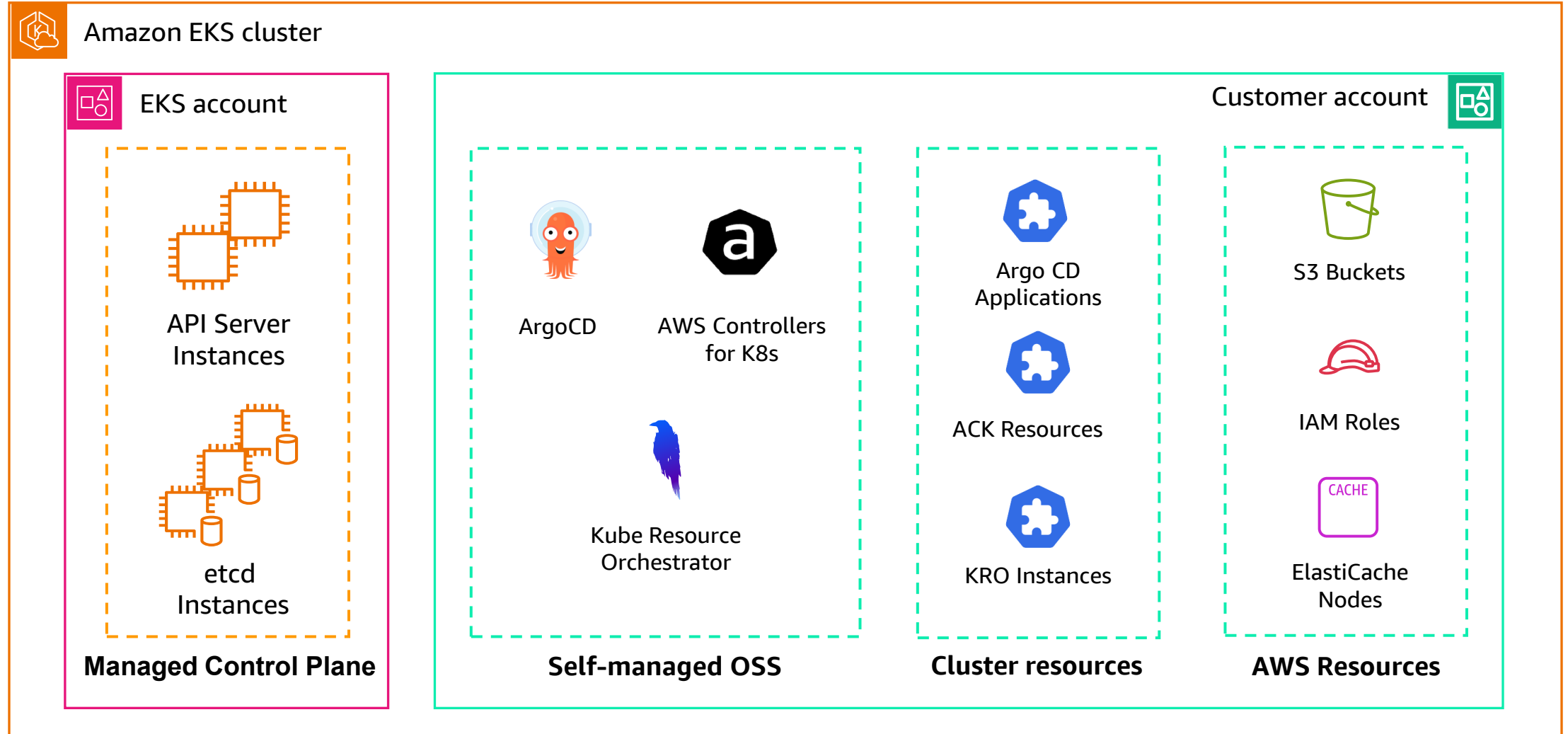


ArgoCD

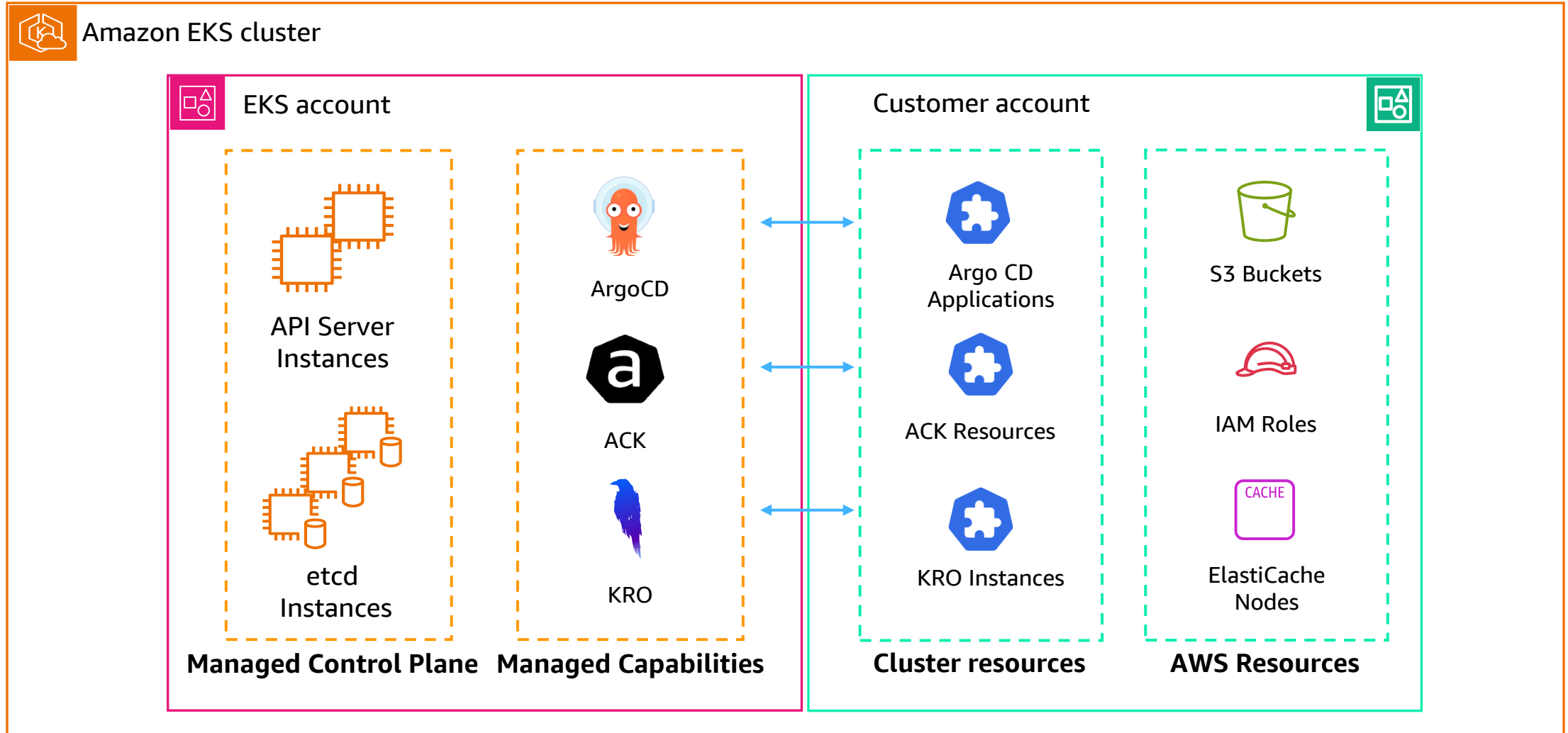


Kube Resource Orchestrator

EKS Capabilitiesを使用しない場合



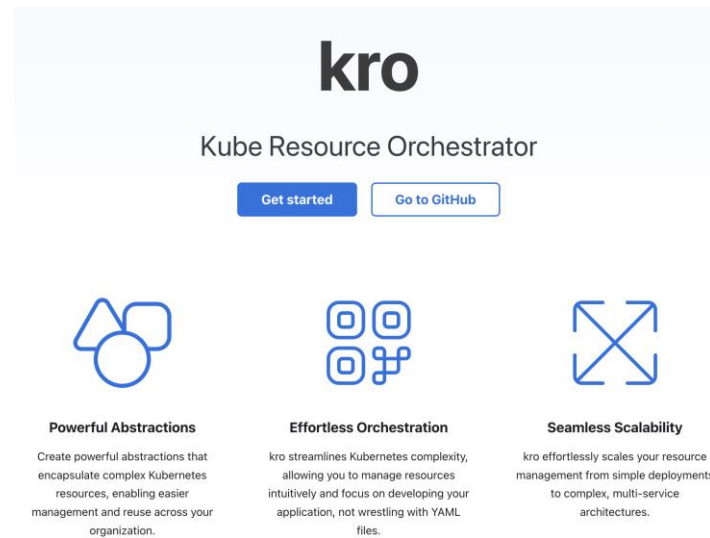
EKS Capabilitiesを使用する場合



Kubernetes Resource Orchestrator (Kro)

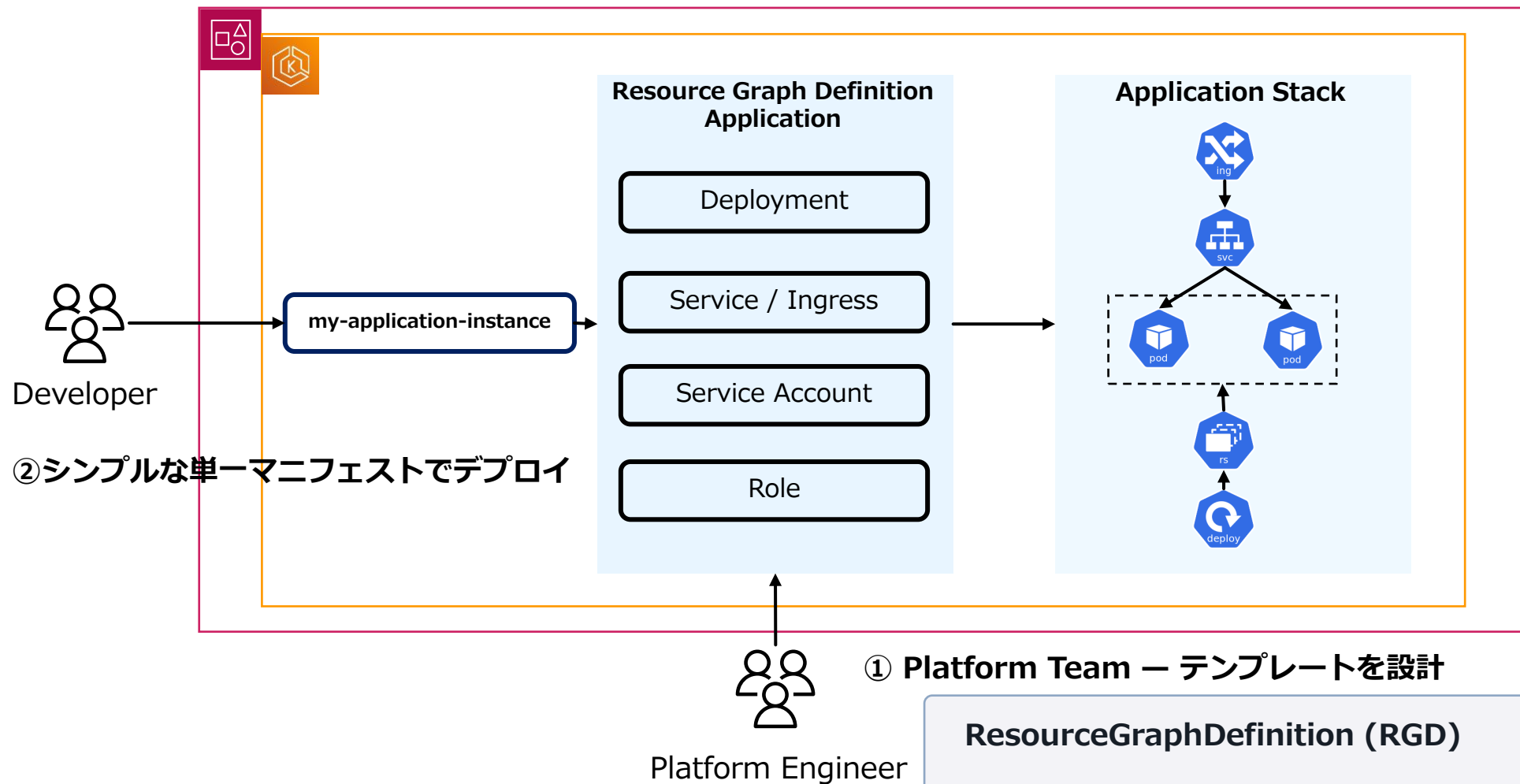
抽象化レイヤを提供するOSS

- ✓ 複数のリソースをまとめて単一のユニットとしてデプロイするための、再利用可能なカスタム Kubernetes API を作成する OSS
- ✓ リソースの依存関係や構成を管理する抽象化レイヤーの提供



kro を利用することで、
「WebApp Instance 1 ファイルで、**k8s** リソースを意識せずアプリをデプロイ」

kroの利用イメージ



ResourceGraphDefinition (RGD)

WebApp や Microservice 等の業務ドメインに沿った CRD を、複数の K8s/AWS リソースの組合せとして定義。依存関係 (作成順序) などここにも集約。

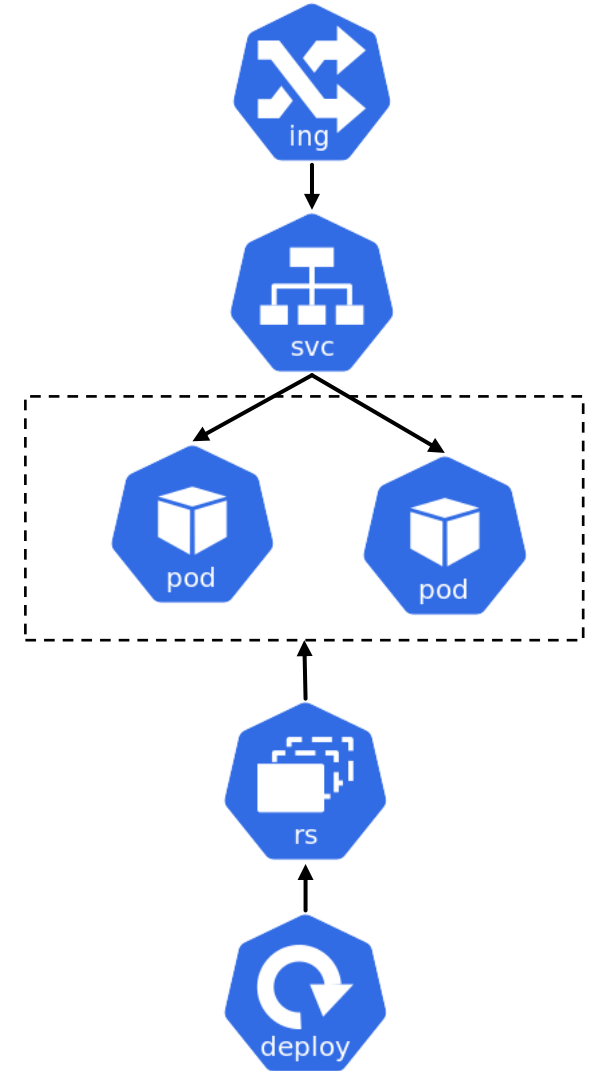
開発者からみたインタフェース



Developer



```
1  apiVersion: kro.run/v1alpha1
2  kind: Application
3  metadata:
4    name: my-application-instance
5  spec:
6    name: my-awesome-app
7    ingress:
8      enabled: true
```



kroのメリット

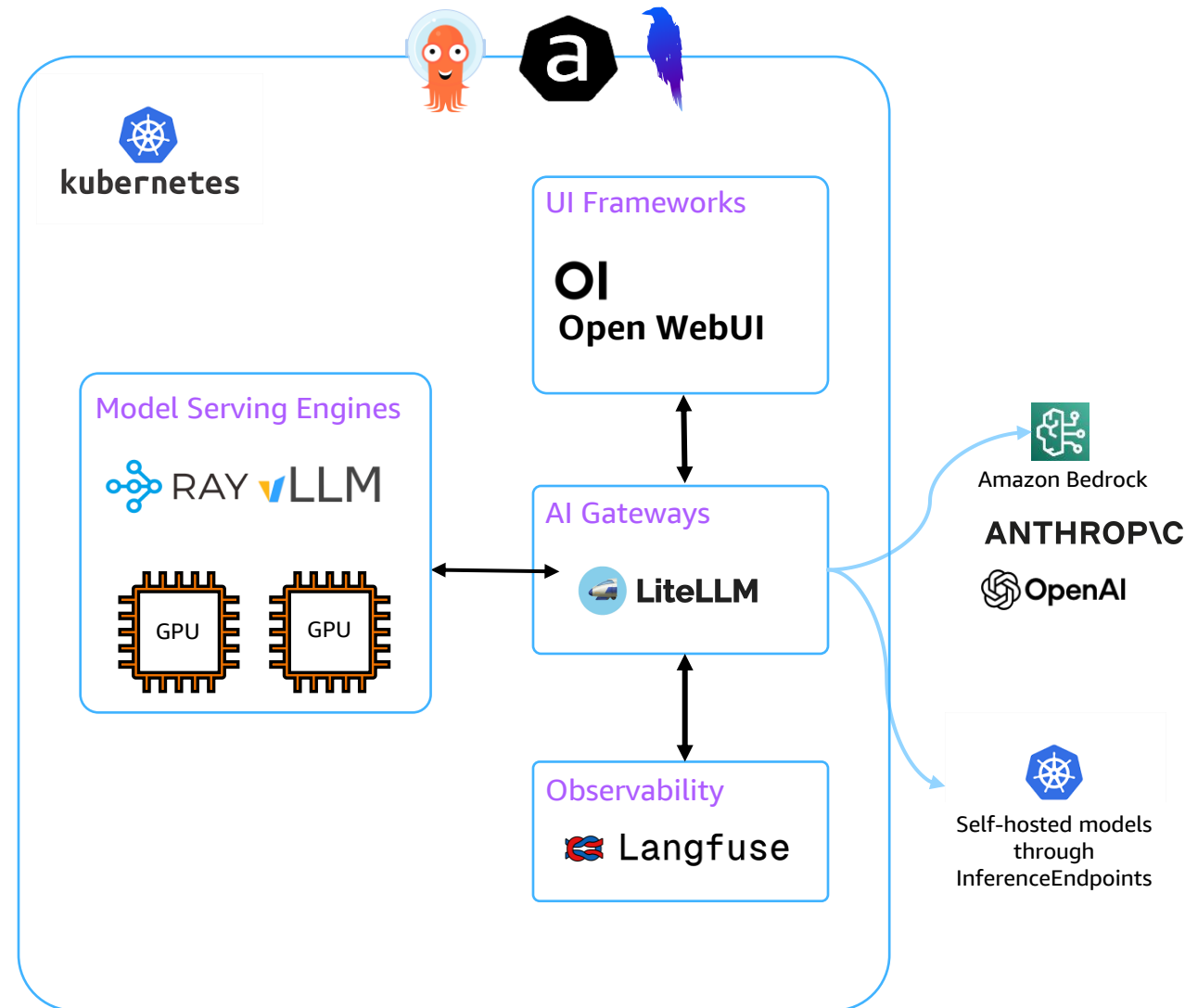
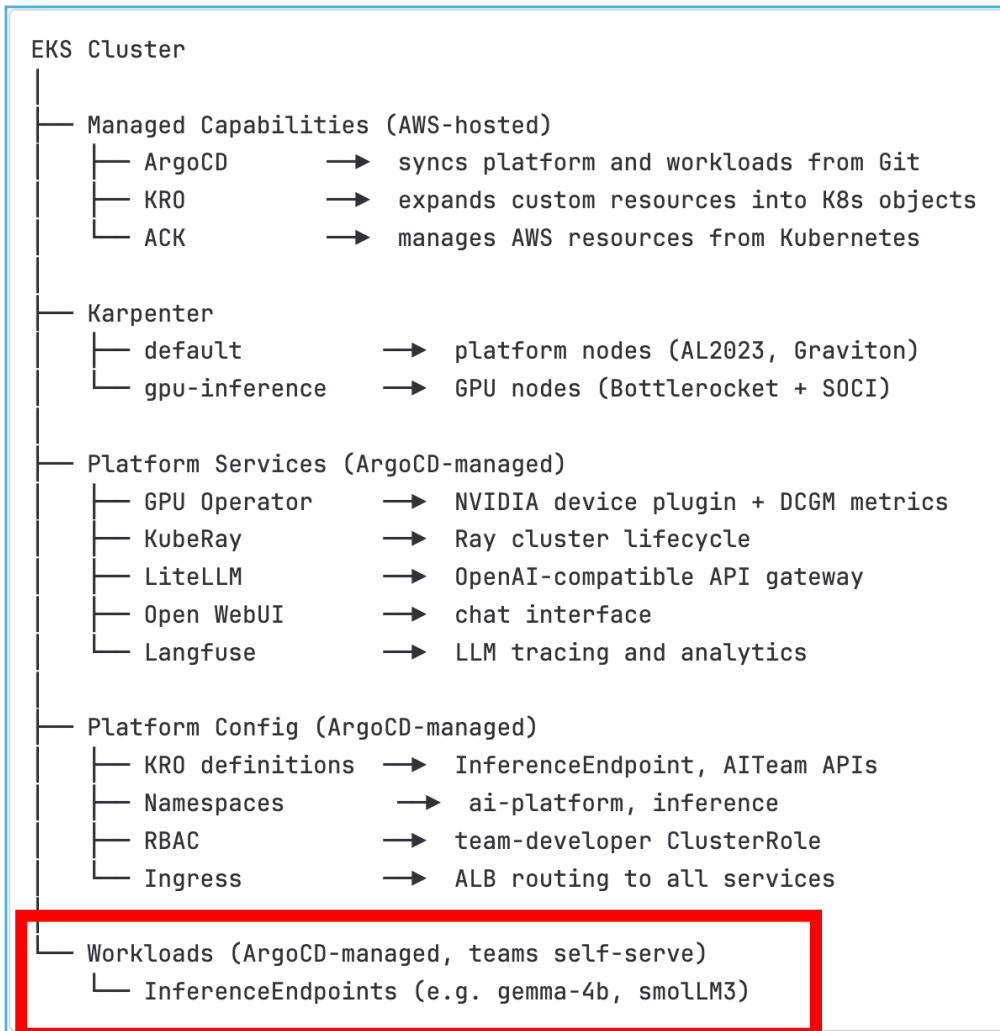
開発チーム が得るメリット

- 複数YAML の暗黙知から解放
- 依存関係の手動管理が不要
- AWS リソース も扱える (ACK 統合)
- Argo CD / GitOps と組み合わせ可能

Platformチーム が得るメリット

- RGD一箇所の変更で全アプリに一括反映
- 開発者に公開するAPIを自分たちで定義・制御できる
- K8sの複雑さを隠蔽しつつ、ガードレールを組み込める
- Helm/Kustomizeのテンプレート肥大化から解放
- 新しい共通パターンの追加がRGD追加だけで完結

デモ アーキテクチャ



Kroによる抽象化により、開発者が意識する箇所は赤枠のみになる

Kro Template

```
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
```

```
metadata:
  name: inference-endpoint
```

```
spec:
```

```
  schema:
```

```
    apiVersion: v1alpha1
```

```
    kind: InferenceEndpoint
```

```
  spec:
```

```
    model: "string" # HuggingFace model ID (e.g. google/gemma-3-4b-it)
    gpuCount: "integer | default=1" # GPUs per worker (tensor_parallel_size)
    minReplicas: "integer | default=1" # Min Ray Serve replicas
    maxReplicas: "integer | default=4" # Max Ray Serve replicas
    workerMemory: "string | default=24Gi" # Memory per GPU worker
    workerCpu: "string | default=4" # CPU per GPU worker
    maxModelLen: "integer | default=8192" # Max sequence length
    rayImage: "string | default=anyscale/ray-llm:2.54.0-py311-cu128"
```

```
resources:
```

```
  # Provides ECR image URI.
```

```
  - id: platformConfig
```

```
  # --- Register model with LiteLLM via API ---
```

```
  - id: litellmregister
```

```
  # --- RayService with vLLM backend ---
```

```
  - id: rayservice
```

```
  # --- CloudWatch Log Group (via ACK) ---
```

```
  - id: loggroup
```

```
# Provides ECR image URI.
```

```
- id: platformConfig
```

```
externalRef:
```

```
  apiVersion: v1
```

```
  kind: ConfigMap
```

```
# --- Register model with LiteLLM via API ---
```

```
- id: litellmregister
```

```
template:
```

```
  apiVersion: batch/v1
```

```
  kind: Job
```

```
...
```

```
args:
```

```
- |
```

```
curl -sf -X POST http://litellm.ai-platform.svc.cluster.local:4000/model/new \
```

```
...
```

```
&& echo " Model registered."
```

```
# --- RayService with vLLM backend ---
```

```
- id: rayservice
```

```
template:
```

```
  apiVersion: ray.io/v1
```

```
  kind: RayService
```

```
...
```

```
  llm_configs:
```

```
    - model_loading_config:
```

```
      model_id: ${schema.spec.model}
```

```
# --- CloudWatch Log Group (via ACK) ---
```

```
- id: loggroup
```

```
template:
```

```
  apiVersion: cloudwatchlogs.services.k8s.aws/v1alpha1
```

```
  kind: LogGroup
```

```
metadata:
```

```
  name: ${schema.metadata.name}-logs
```

```
spec:
```

```
  name: /ai-platform/models/${schema.metadata.name}
```

このRGDはPFチーム側で管理する

Kroによる抽象化

- ✓ Kroはモデルのデプロイを、たった1つの InferenceEndpoint APIに抽象化する
- ✓ 開発チームは Ray・vLLM・関連する k8s/AWSサービスを意識する必要がない
- ✓ ユースケースに応じてテンプレートを拡張できる

! gemma-4b.yaml ×

platform > workloads > ! gemma-4b.yaml

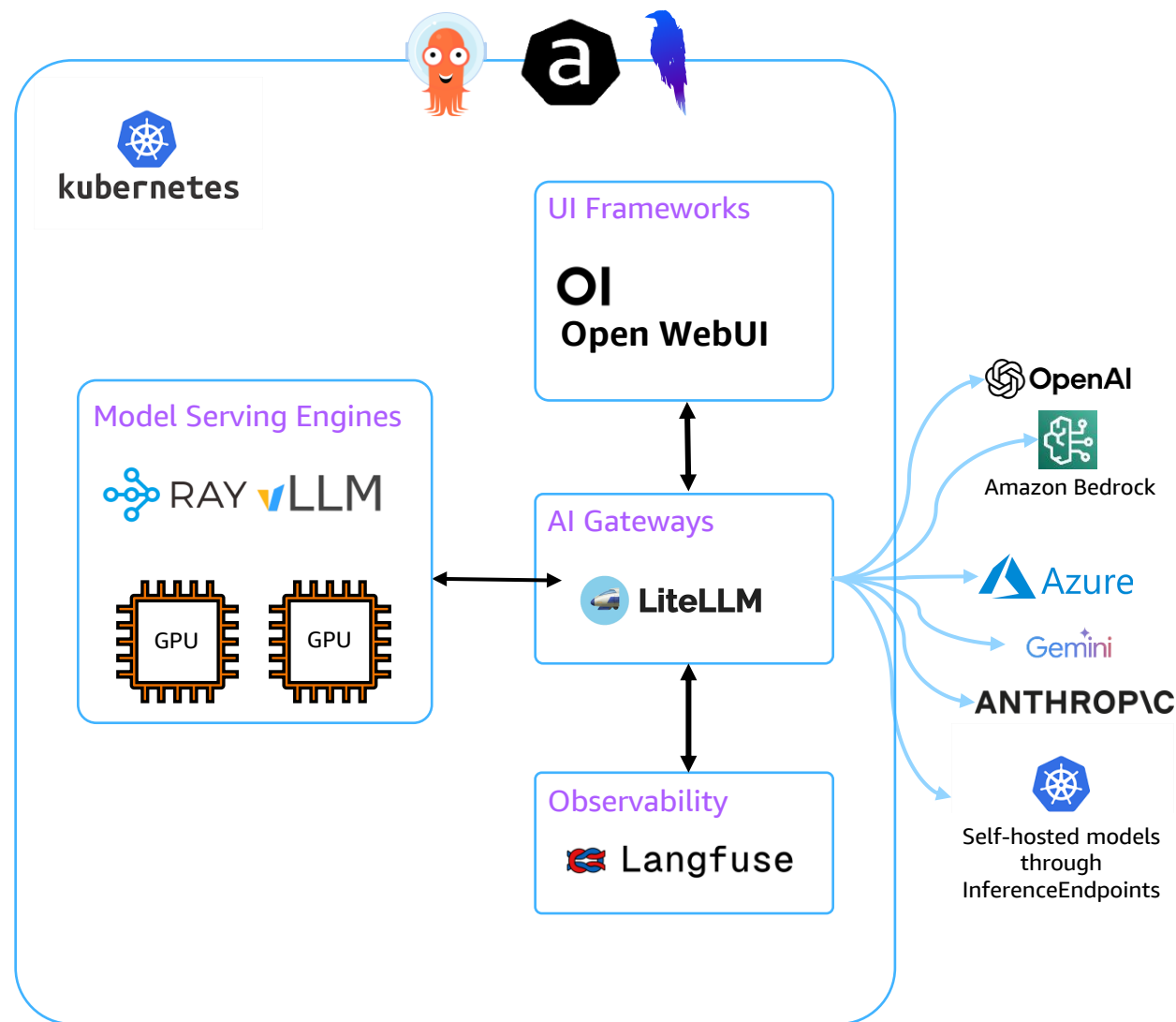
```
1  apiVersion: kro.run/v1alpha1
2  kind: InferenceEndpoint
3  metadata:
4    name: gemma-4b
5    namespace: inference
6  spec:
7    model: "google/gemma-3-4b-it"
8    gpuCount: 1
9    minReplicas: 1
10   maxReplicas: 2
11
```

ゴールデンパス

開発者は以下の `InferenceEndpoint` を宣言して `git push` する。

あとはプラットフォームがGPUをプロビジョニングし、モデルを配信（例：Ray Serve + vLLM）して、セルフサービスでアプリを公開します。

```
! gemma-4b.yaml ×
platform > workloads > ! gemma-4b.yaml
1  apiVersion: kro.run/v1alpha1
2  kind: InferenceEndpoint
3  metadata:
4    name: gemma-4b
5    namespace: inference
6  spec:
7    model: "google/gemma-3-4b-it"
8    gpuCount: 1
9    minReplicas: 1
10   maxReplicas: 2
11
```



aws

Search

[Option+S]

39+

Europe (Frankfurt)

Radu (8020-1929-9867)

AdminRole/rdobrine-lsengard

Amazon Elastic Kubernetes Service

Clusters

ai-platform-radu-test

Amazon Elastic Kubernetes Service

Dashboard

Clusters

Settings

Dashboard settings

Console settings

Amazon EKS Anywhere

Enterprise Subscriptions

Related services

Amazon ECR

AWS Batch

Documentation

EKS workshops

ai-platform-radu-test

Refresh

Delete cluster

Monitor cluster

Cluster info

Status

Active

Kubernetes version

1.35

Support period

Standard support until March 27, 2027

Provider

EKS

Cluster health

0

Upgrade insights

5

Node health issues

0

Capability issues

0

Overview

Resources

Compute

Networking

Add-ons

Capabilities

Access

Observability

Update history & backups

Tags

Details

API server endpoint

https://A9CAB8498A82AF5EADA25837412ADA06.gr7.eu-central-1.eks.amazonaws.com

OpenID Connect provider URL

https://oidc.eks.eu-central-1.amazonaws.com/id/A9CAB8498A82AF5EADA25837412ADA06

Created

February 28, 2026, 11:47 (UTC+02:00)

Certificate authority

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSOtLS0tck1JSURCVENDQWUyZ0F3SUJBZ0JYkFzbF3QmJOW93RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdz

Cluster IAM role ARN

arn:aws:iam::802019299867:role/ai-platform-radu-test-cluster

View in IAM

Cluster ARN

arn:aws:eks:eu-central-1:802019299867:cluster/ai-platform-radu-test

Platform version

eks.5

EKS Auto Mode

EKS automates routine cluster tasks for compute, storage, and networking to meet application compute needs.

Disabled

Manage

Kubernetes version settings

Configure your cluster to either enter extended support or automatically upgrade when standard support ends.

Upgrade policy

Extended support

Manage

CloudShell

Feedback

© 2026, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

Cookie preferences



Thank you!

Please complete the
session survey →



Appendix

ACK Capability – All AWS Services Controllers


[Documentation](#)
[API Reference](#)

[Documentation for information on how we version our releases](#)
[AWS Service Catalog](#)




Introduction

- Overview
- Deprecation Policy
- How it Works
- Services
- Release Phases

> Getting Started

- Tutorials
- Contributor Docs
- Discussion

Maintenance Phase

GENERAL AVAILABILITY

PREVIEW

Services

54

4

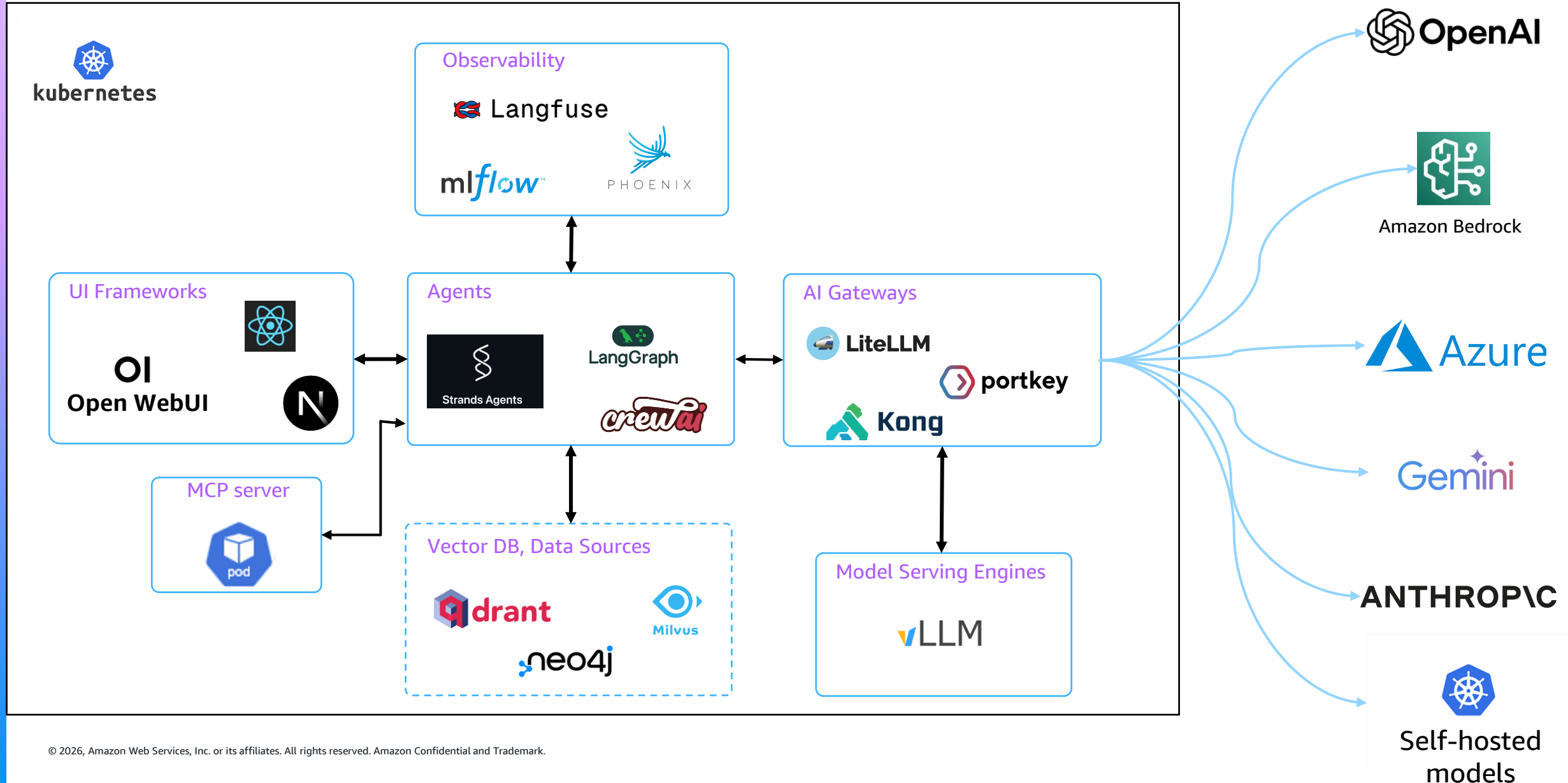
AWS Service	Project Stage	Maintenance Phase	Latest Version
Amazon ACM-PCA	RELEASED	GENERAL AVAILABILITY	1.2.1
Amazon ACM	RELEASED	GENERAL AVAILABILITY	1.3.3
Amazon Prometheus Service	RELEASED	GENERAL AVAILABILITY	1.4.1
Amazon API Gateway	RELEASED	GENERAL AVAILABILITY	1.4.1
Amazon ApiGatewayV2	RELEASED	GENERAL AVAILABILITY	1.2.1
Amazon Application Auto Scaling	RELEASED	GENERAL AVAILABILITY	1.2.1
AWS App Runner	PROPOSED	NONE	n/a
Amazon Athena	RELEASED	GENERAL AVAILABILITY	1.2.1
AWS Auto Scaling Plans	PROPOSED	NONE	n/a
Amazon Auto Scaling	PROPOSED	NONE	n/a
Amazon Agents for Amazon Bedrock	RELEASED	GENERAL AVAILABILITY	1.1.1
Amazon Bedrock	RELEASED	GENERAL AVAILABILITY	1.1.1
Amazon CloudFront	RELEASED	GENERAL AVAILABILITY	1.3.1
Amazon CloudTrail	RELEASED	GENERAL AVAILABILITY	1.2.1
Amazon CodeArtifact	RELEASED	GENERAL AVAILABILITY	1.2.1
Amazon Cognito Identity Provider	RELEASED	GENERAL AVAILABILITY	1.2.1
Amazon DocDB	RELEASED	GENERAL AVAILABILITY	1.3.0
Amazon DynamoDB	RELEASED	GENERAL AVAILABILITY	1.7.1

On this page

[Amazon ACM-PCA](#)
[Amazon ACM](#)
[Amazon Prometheus Service](#)
[Amazon API Gateway](#)
[Amazon ApiGatewayV2](#)
[Amazon Application Auto Scaling](#)
[AWS App Runner](#)
[Amazon Athena](#)
[AWS Auto Scaling Plans](#)
[Amazon Auto Scaling](#)
[Amazon Agents for Amazon Bedrock](#)
[Amazon Bedrock](#)
[Amazon CloudFront](#)
[Amazon CloudTrail](#)
[Amazon CodeArtifact](#)
[Amazon Cognito Identity Provider](#)
[Amazon DocDB](#)
[Amazon DynamoDB](#)
[Amazon EC2](#)
[Amazon ECR Public](#)
[Amazon ECR](#)
[Amazon ECS](#)
[Amazon EKS](#)
[Amazon ElastiCache](#)
[Amazon EFS](#)
[Amazon Elastic Load Balancing v2](#)
[Amazon SES](#)
[Amazon EMR Containers](#)

<https://aws-controllers-k8s.github.io/community/docs/community/services/>

AI Architecture using open-source frameworks



Self-service Model Inference



```
! gemma-4b.yaml ×
platform > workloads > ! gemma-4b.yaml
1  apiVersion: kro.run/v1alpha1
2  kind: InferenceEndpoint
3  metadata:
4    name: gemma-4b
5    namespace: inference
6  spec:
7    model: "google/gemma-3-4b-it"
8    gpuCount: 1
9    minReplicas: 1
10   maxReplicas: 2
11
```

- 1 Commit&Push
- 2 Interact



Getting started

```
capabilities = {  
  gitops = true # ArgoCD - requires AWS Identity Center (see capabilities_config)  
  kro     = true # Kube Resource Orchestrator  
  ack     = true # AWS Controllers for Kubernetes  
}  
  
# Required when gitops = true  
capabilities_config = {  
  argocd_idc_instance_arn = "arn:aws:sso:::instance/ssoins-XXXXXXXXXX" // REPLACE with your IdC instance ARN  
  argocd_idc_region       = "us-east-1"                               // Region where Identity Center is configured  
  argocd_rbac_mappings = [  
    {  
      role = "ADMIN"  
      identities = [  
        { id = "REPLACE-WITH-SSO-USER-ID", type = "SSO_USER" }  
      ]  
    }  
  ]  
}
```

	Managed capabilities ▲	Capability name ▼	Status ▼
☐	AWS Controllers for Kubernetes (ACK)	ack	✔ Active
☐	Argo CD	argocd	✔ Active
☐	Kube Resource Orchestrator (KRO)	kro	✔ Active