#EKSMatsuri

Writing Custom Controllers

– Extends Kubernetes APIs for the unified experience

Tori Hara (@toricls)

Sr. Developer Advocate
Containers Product, Amazon Web Services

Apr. 30, 2020

Tori Hara (@toricls)

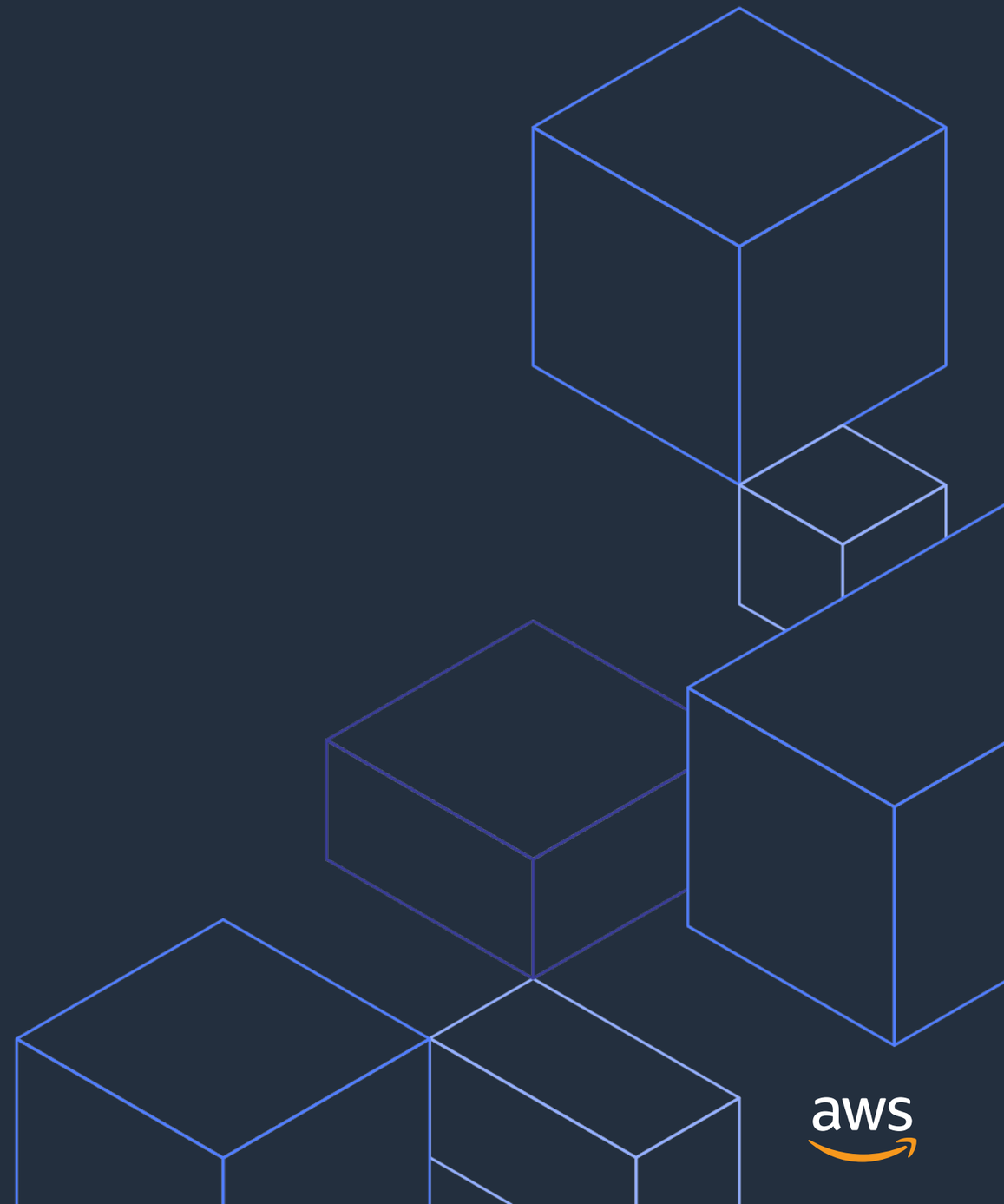
Sr. Developer Advocate
Containers Product, Amazon Web Services

ERP パッケージベンダー R&D チーム SDE
➡ UI 自動テスト SaaS
➡ クラウド利用の SI + MSP にて、コンテナやサーバーレスによる設計・開発・運用
　Web 技術利用のゲームやビジネスアプリケーション開発、ML/DL 環境構築運用など
➡ Sr. Containers Specialist SA, AWS Japan
➡ 現ロール

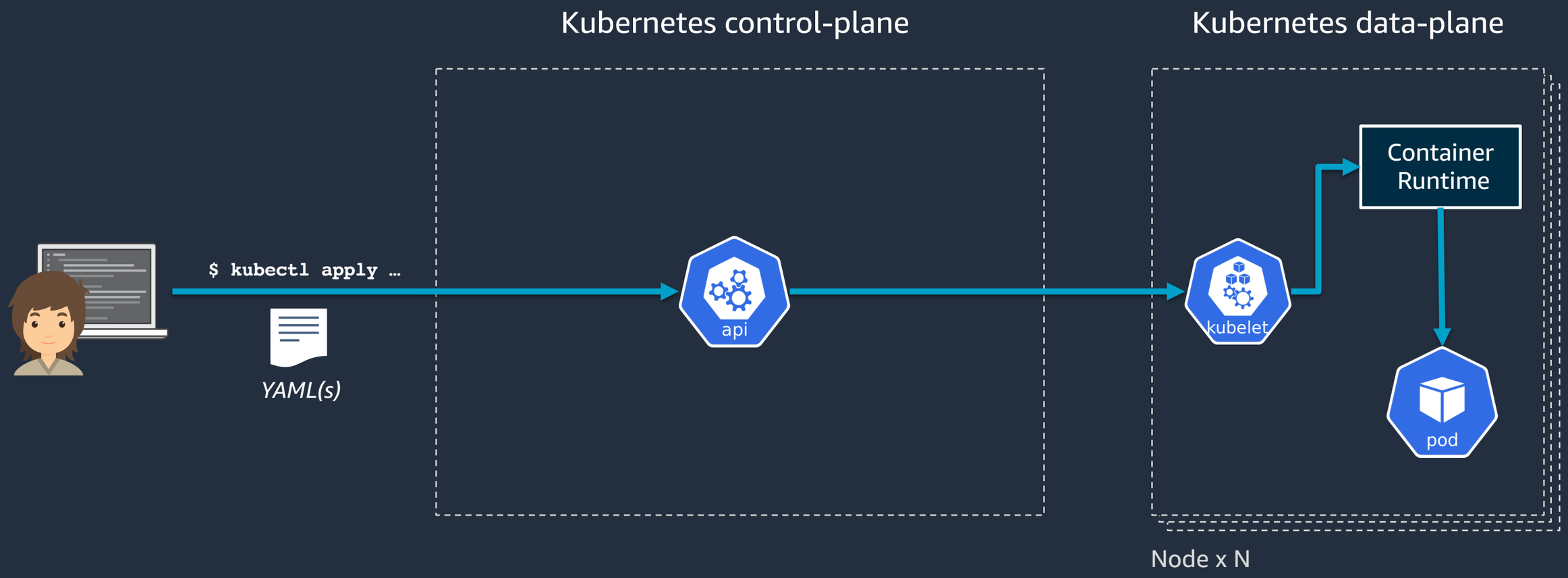
AWS Fargate / AWS Lambda が好きです

aws

Kubernetes の価値

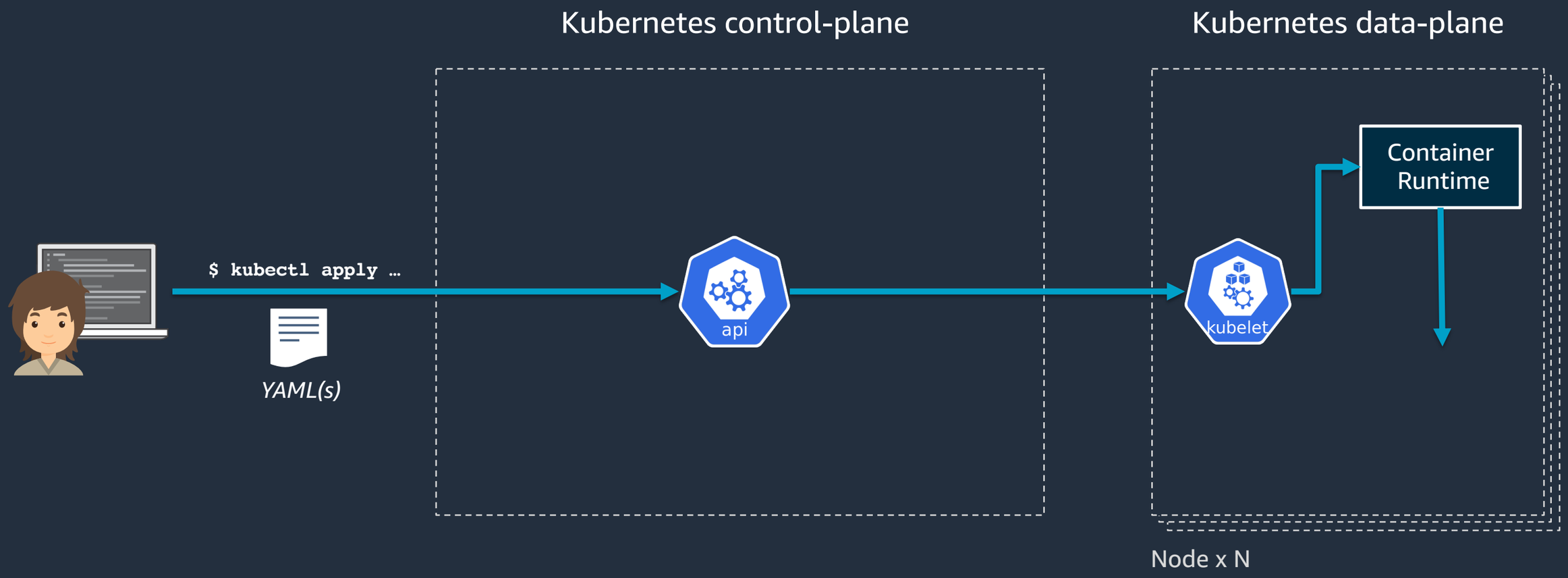


Kubernetes の宣言的デプロイメントモデル (超概略)



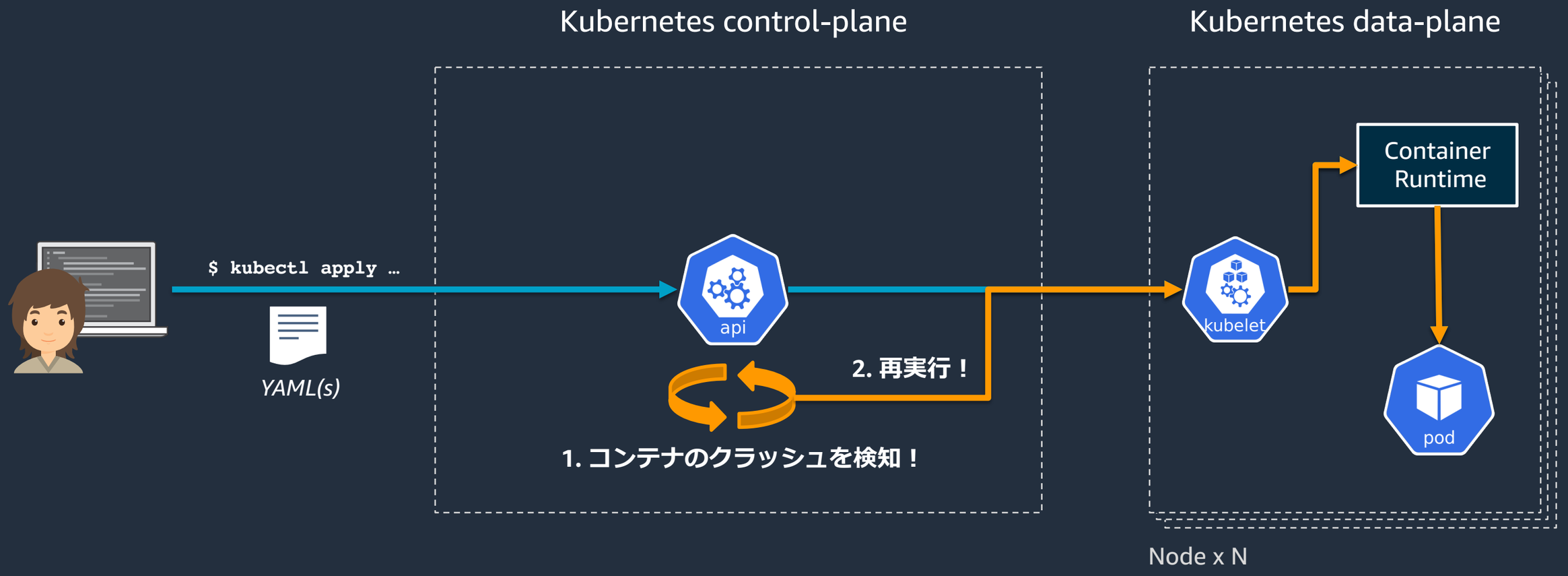
aws

Kubernetes の宣言的デプロイメントモデル (超概略)

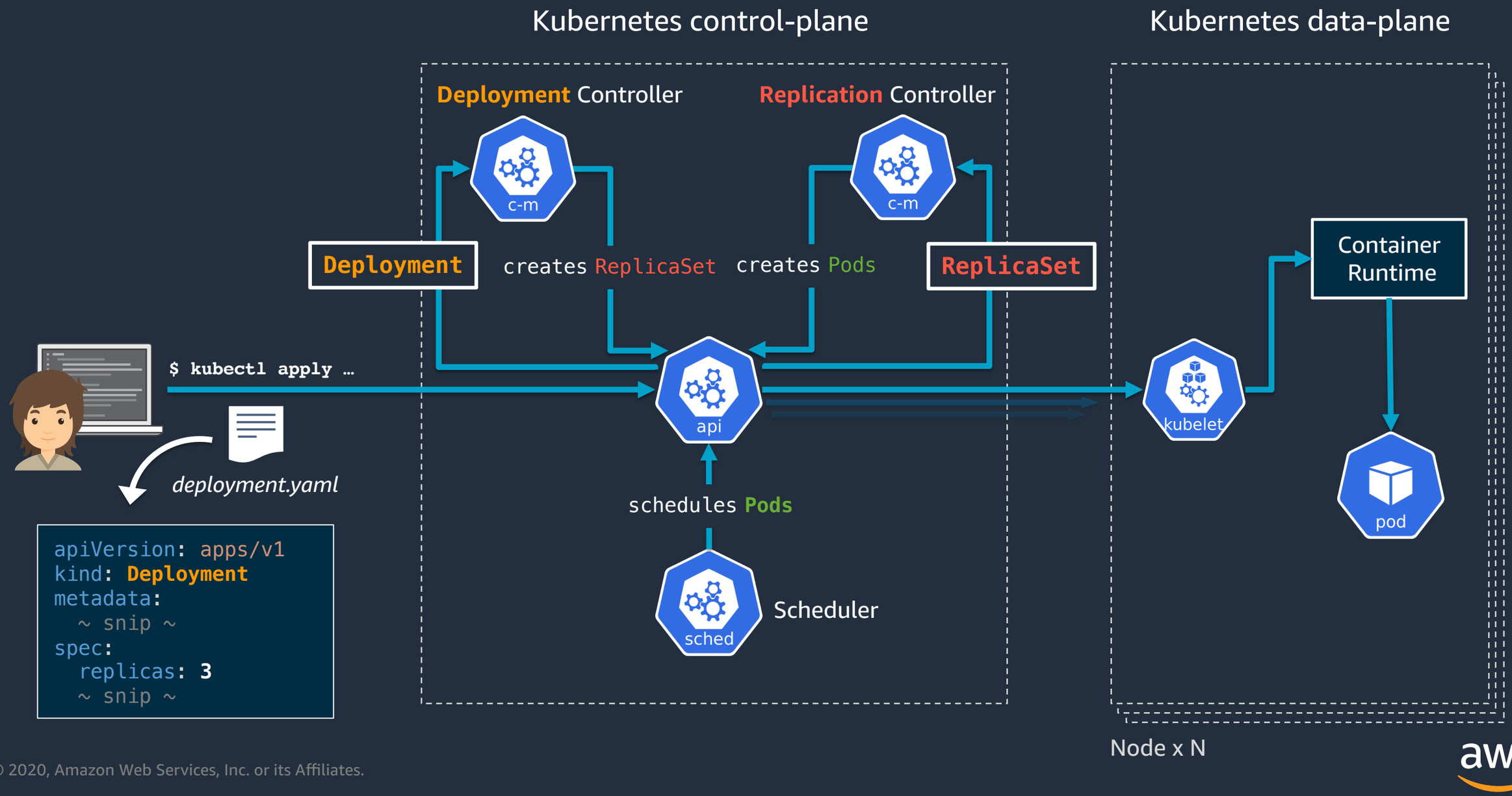


aws

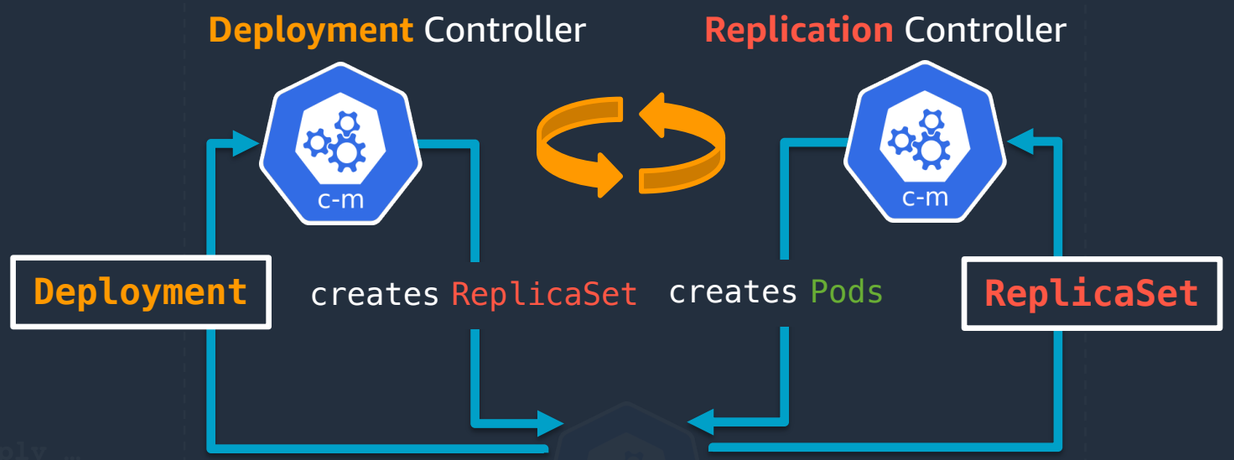
Kubernetes の宣言的デプロイメントモデル (超概略)



Kubernetes の宣言的デプロイメントモデル (概略)



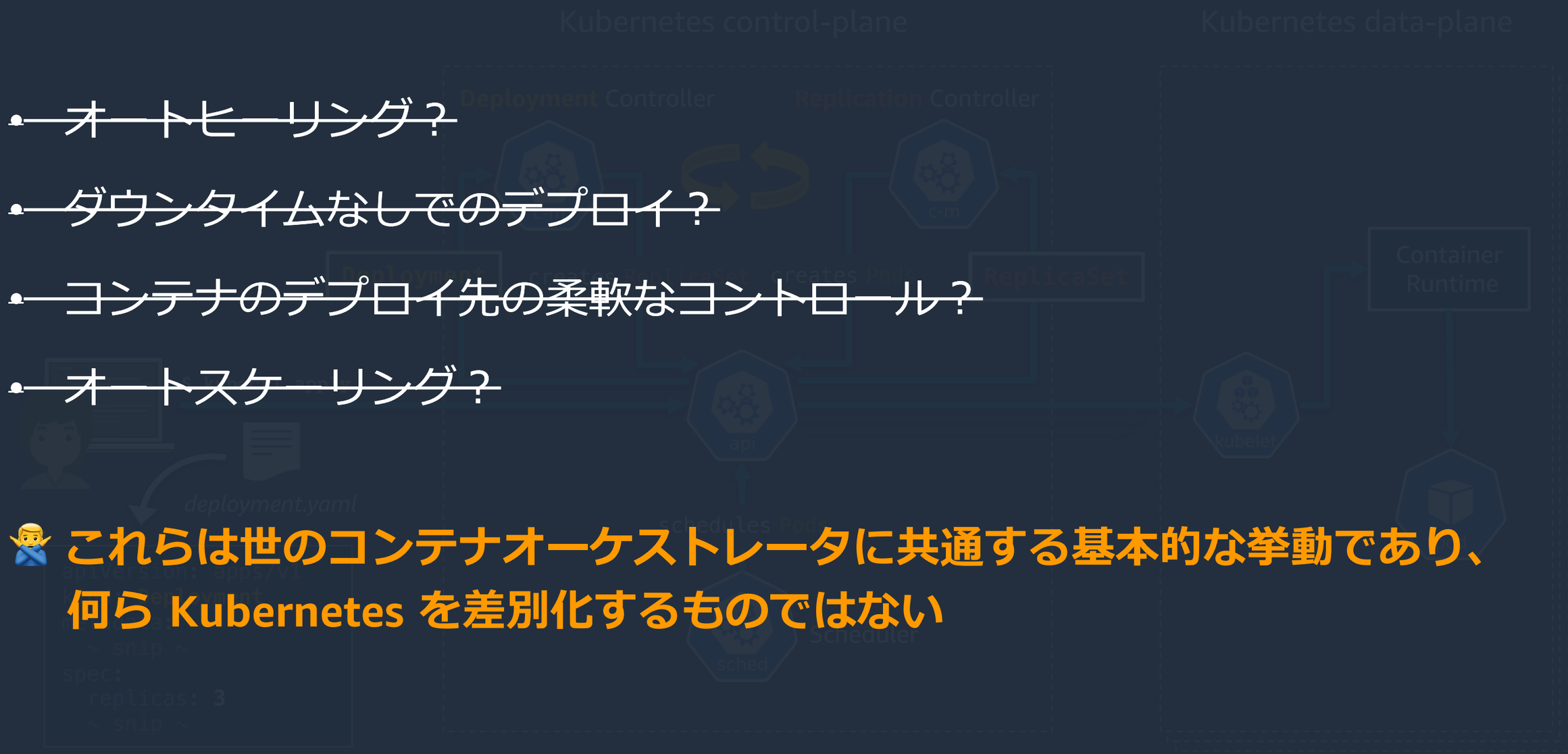
Kubernetes の宣言的デプロイメントモデル (概略)



- "**control loops**" または "**reconcile loops**" と呼ばれる
- プログラミング言語におけるループの概念と近く、このループの中で宣言と現実の状態差異を検出し、収束させるロジックを常に実行している

Kubernetes を差別化しているものは何か

- ~~オートヒーリング？~~
- ~~ダウンタイムなしでのデプロイ？~~
- ~~コンテナのデプロイ先の柔軟なコントロール？~~
- ~~オートスケーリング？~~



🤦‍♂️ **これらは世のコンテナオーケストレータに共通する基本的な挙動であり、何ら Kubernetes を差別化するものではない**

aws

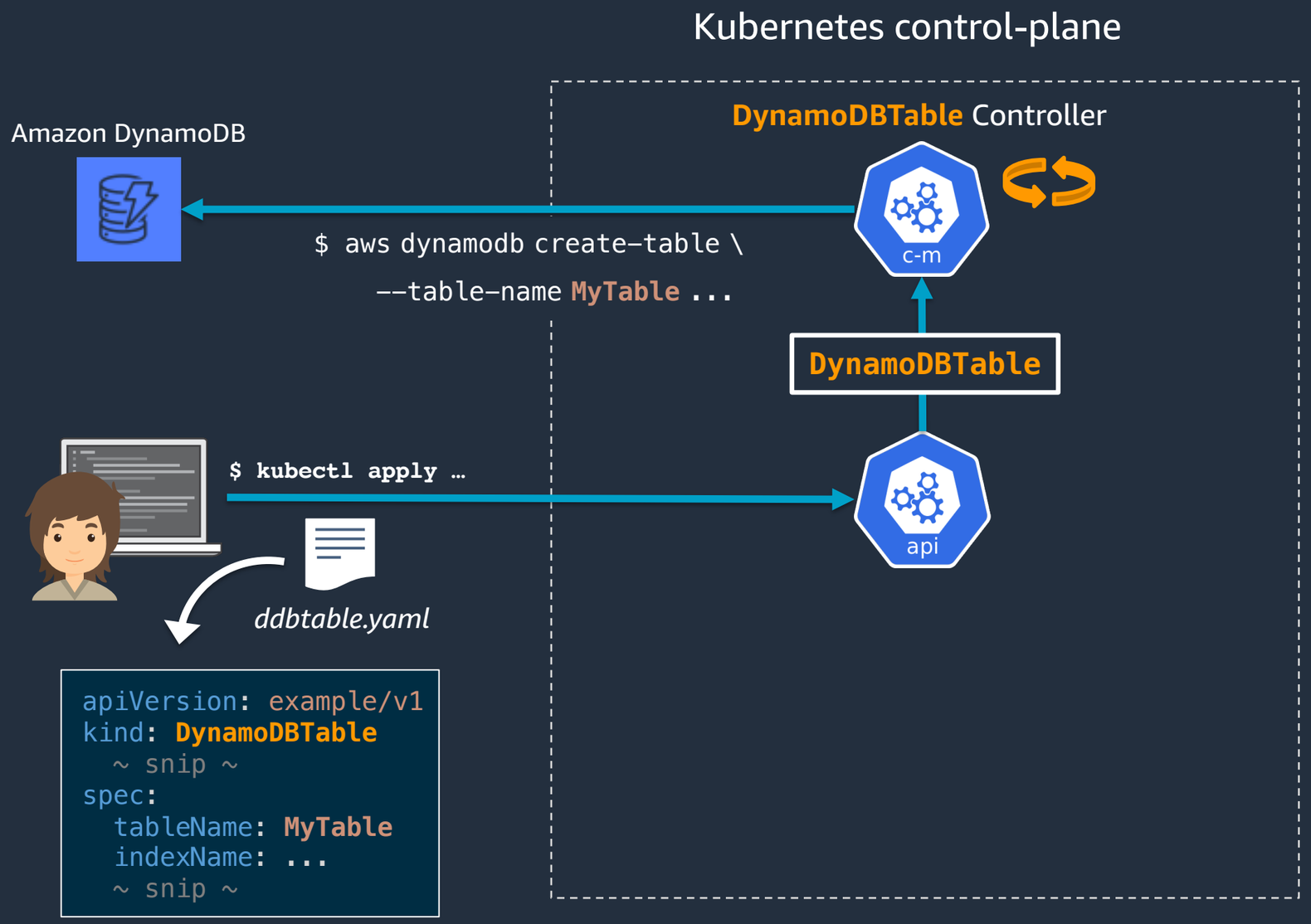
Kubernetes のコンテナオーケストレータとしての重要な価値

高い拡張可能性

一貫性のある
デプロイメントモデル

aws

Kubernetes の一貫性ある宣言的デプロイメントモデル (概略)



Kubernetes の高い拡張性は どのようにして実現されているのか

aws

Kubernetes の拡張性

- コンテナレベルの拡張仕様のサポート
    - Container Network Interface (CNI), Container Storage Interface(CSI)
- コンテナランタイムのプラグインインターフェース
    - Container Runtime Interface (CRI)
- Validating/Mutating Admission Webhook
- カスタムスケジューラの実装と利用
- …
- **カスタム実装による Kubernetes API のプラガブルな拡張** (Today’s topic)

aws

Kubernetes API Overview

```
$ curl https://<YOUR_KUBERNETES_API_ENDPOINT>/

{
  "paths": [
    "/api",
    "/api/v1",
      ~ snip ~
    "/apis/apps/v1",
      ~ snip ~
    "/apis/networking.k8s.io/v1",
      ~ snip ~
    "/logs",
    "/metrics",
    "/version"
  ]
}
```

※ デモ目的で RBAC を無効化したローカル Kubernetes クラスタにて実行しています.
普段ご利用の Kubernetes 環境における RBAC 無効化は**強く非推奨**です.

aws

Kubernetes API Overview

```
$ curl https://<YOUR_KUBERNETES_API_ENDPOINT>/api/v1

{
  "kind": "APIResourceList",
  "groupVersion": "v1",
  "resources": [
    {
      "name": "pods",
      "kind": "Pod",
      "shortNames": ["po"]
      ~ snip ~
    },
    {
      ~ snip ~
  ]
}
```

※ デモ目的で RBAC を無効化したローカル Kubernetes クラスタにて実行しています.
普段ご利用の Kubernetes 環境における RBAC 無効化は**強く非推奨**です.

aws

Kubernetes API Overview

```
$ curl https://<YOUR_KUBERNETES_API_ENDPOINT>/api/v1/pods

{ ~ snip ~
  "metadata": {
    "selfLink": "/api/v1/pods",
    "resourceVersion": "590660"
  },
  "items": [{
      "metadata": {
        "name": "nginx-deployment-7b758584f7-cqb6b",
          ~ snip ~
    } {
      "metadata": {
        "name": "coredns-5d4dd4b4db-8xdrn",
          ~ snip ~
]}
```

※ デモ目的で RBAC を無効化したローカル Kubernetes クラスタにて実行しています.
普段ご利用の Kubernetes 環境における RBAC 無効化は**強く非推奨**です.

aws

Kubernetes API Overview

```
$ kubectl get pods --all-namespaces -v=7

I0428 19:34:42   Config loaded from file:  /<YOUR_HOME>/.kube/config
I0428 19:38:05   GET https://<YOUR_KUBERNETES_API_ENDPOINT>/api/v1/pods

 ~ snip ~

I0428 19:38:05   Response Status: 200 OK in 33 milliseconds

NAMESPACE     NAME                                  READY   STATUS    ...
default       nginx-deployment-7b758584f7-cqb6b     1/1     Running   ...
kube-system   coredns-5d4dd4b4db-8xdrn              1/1     Running   ...

 ~ snip ~
```

- Kubernetes API = HTTP での Kubernetes リソースの操作を提供
- kubectl = Kubernetes API と喋ることができるクライアント

aws

Kubernetes の宣言的デプロイメントモデル (概略)

再掲

- 新規作成 - POST https://<K8s_API>/apis/apps/v1/namespaces/default/deployments
- 更新 - PATCH https://<K8s_API>/apis/apps/v1/namespaces/default/deployments/<resource_name>
- YAML から Request Body が生成される


```
apiVersion: example/v1
kind: Deployment
metadata:
  ~ snip ~
spec:
  replicas: 3
  ~ snip ~
```


aws

拡張された Kubernetes API の様子

```
$ curl https://<YOUR_KUBERNETES_API_ENDPOINT>/

{
  "paths": [
    "/api",
    "/api/v1",
      ~ snip ~
    "/apis/apps/v1",
      ~ snip ~
    "/apis/my-resource.toris.io",
    "/apis/my-resource.toris.io/v1",
      ~ snip ~
    "/metrics",
    "/version"
  ]
}
```

※ デモ目的で RBAC を無効化したローカル Kubernetes クラスタにて実行しています.
普段ご利用の Kubernetes 環境における RBAC 無効化は**強く非推奨**です.

aws

拡張された Kubernetes API の様子

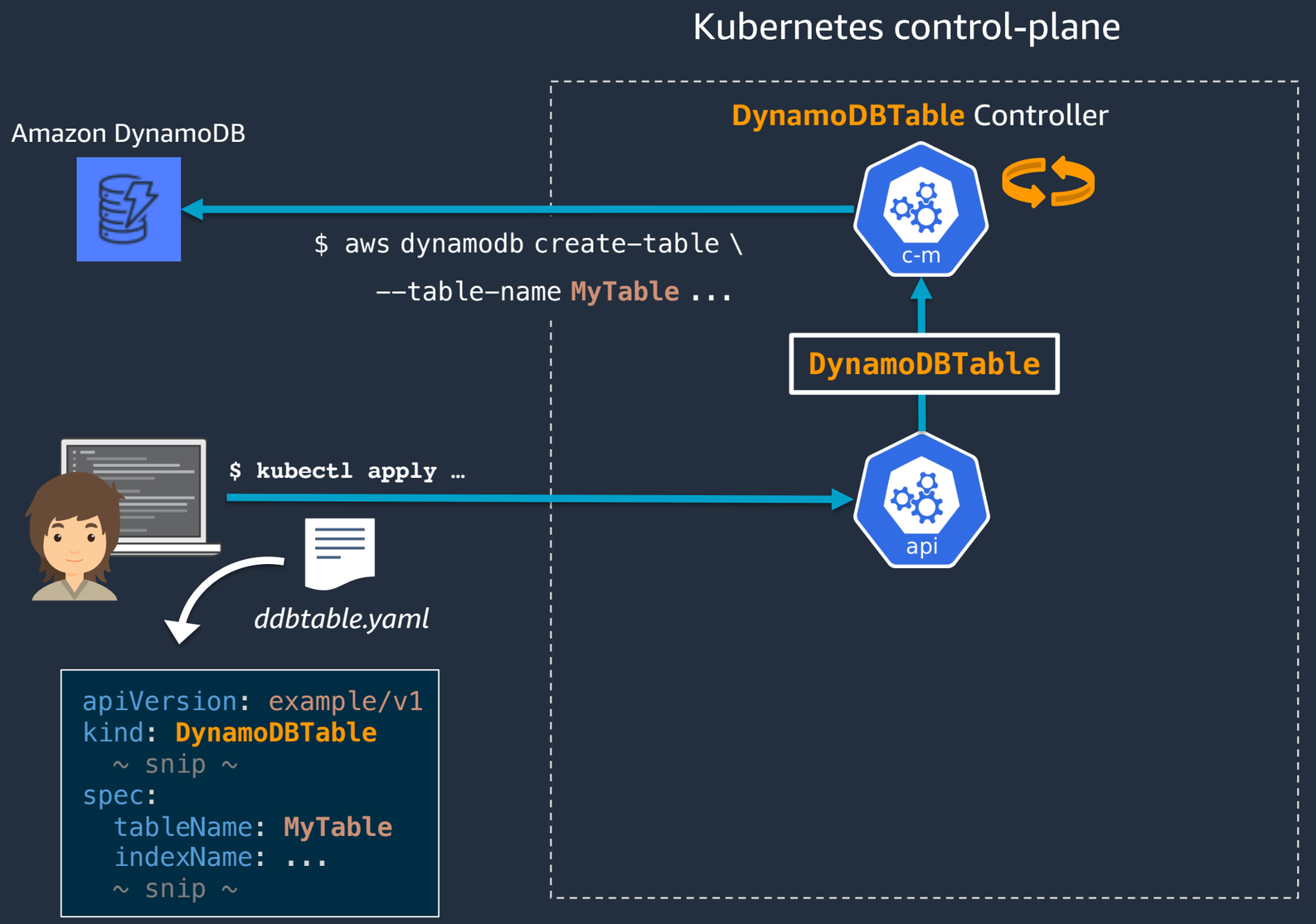
```
$ kubectl get mr --all-namespaces -v=7

I0428 19:34:42   Config loaded from file:  /<YOUR_HOME>/.kube/config
I0428 19:38:05   GET https://<YOUR_KUBERNETES_API_ENDPOINT>/apis/my-resource.toris.io/v1

 ~ snip ~

I0428 19:38:05   Response Status: 200 OK in 30 milliseconds

NAMESPACE        NAME                      AGE        ...
hey-yo           my-super-duper-resource   1h13m      ...
 ~ snip ~
```

- 拡張された Kubernetes API も標準の Kubernetes API 同様に kubectl で操作できる
- Kubernetes 標準リソース(Pods, Deployments...)だけではなく独自のリソースを定義し、それらを kubectl + YAML で管理できる

aws

Kubernetes の一貫性ある宣言的デプロイメントモデル (概略)

再掲



Kubernetes API の代表的な拡張実装方法

1. Aggregation Layer (API aggregation / AA)
    - *The aggregation layer allows Kubernetes to be extended with additional APIs, beyond what is offered by the core Kubernetes APIs.* [1]
    - (参考訳) アグリゲーションレイヤーによって API の追加と Kubernetes の拡張が可能となり、Kubernetes コア API が提供する機能性を超えた機能性を実現できます

2. Custom Resources (CustomResourceDefinition + Custom Controller, CRD)
    - *Custom resources are extensions of the Kubernetes API.* [2]
    - (参考訳) カスタムリソースは Kubernetes API の拡張です

[1] https://kubernetes.io/ja/docs/concepts/extend-kubernetes/api-extension/apiserver-aggregation/
[2] https://kubernetes.io/ja/docs/concepts/extend-kubernetes/api-extension/custom-resources/

aws

Kubernetes API の代表的な拡張実装方法

1. Aggregation Layer (API aggregation / AA)
    - *The aggregation layer allows Kubernetes to be extended with additional APIs, beyond what is offered by the core Kubernetes APIs.* [1]
    - (参考訳) アグリゲーションレイヤーによって API の追加と Kubernetes の拡張が可能となり、Kubernetes コア API が提供する機能性を超えた機能性を実現できます

2. Custom Resources (CustomResourceDefinition + Custom Controller, CRD)
    - *Custom resources are extensions of the Kubernetes API.* [2]
    - (参考訳) カスタムリソースは Kubernetes API の拡張です

[1] https://kubernetes.io/ja/docs/concepts/extend-kubernetes/api-extension/apiserver-aggregation/
[2] https://kubernetes.io/ja/docs/concepts/extend-kubernetes/api-extension/custom-resources/

aws

Kubernetes API の代表的な拡張実装方法の違い

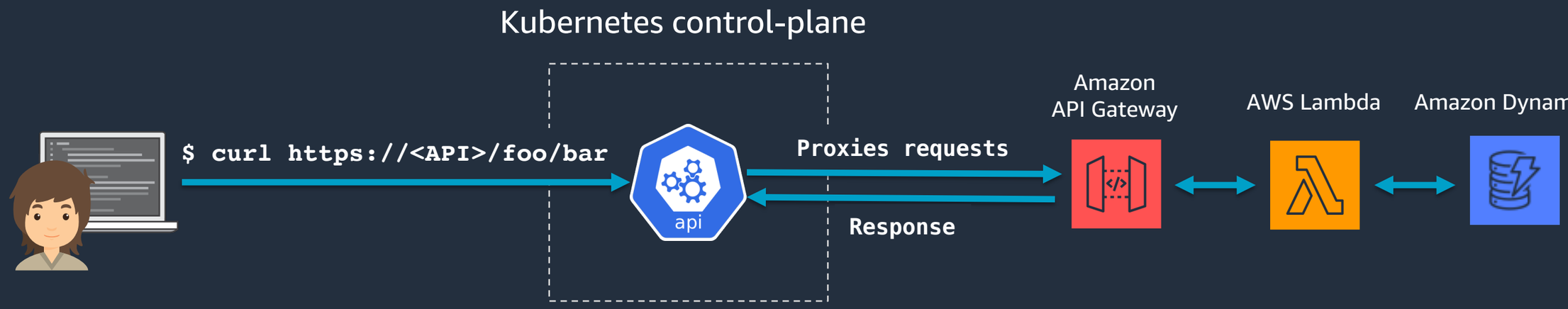
1. Aggregation Layer (API aggregation / AA)
    - Kubernetes の **RESTful API としてのリソース**を任意に追加可能
    - とにかく自由度が高い

2. Custom Resources (CustomResourceDefinition, CRD)
    - **Kubernetes としてのリソース**を任意に追加可能
        - e.g. 前述の “kind: DynamoDBTable”
    - Kubernetes リソースは Kubernetes RESTful API のリソースとして表現されるため、結果として API リソースが追加される

※ Custom resource だけでは単にデータを入れる箱にしかならないため、多くのケースで Custom controller をビジネスロジック実装のために併用

aws

1. Aggregation Layer による Kubernetes API 拡張例



- 拡張した Kubernetes API リソースの永続化ストレージに etcd 以外のデータソースを利用したいケース
- 標準的な Kubernetes リソースには存在しないオペレーション(e.g. *logs*, *exec*)を定義したいケース
- 外に独自のコントローラ(control loop)を実装することも多い

※　実際のところ API aggregation ではコードを書けばほぼなんでもできるため、なんなら Custom Resource との組み合わせによる実装も可能

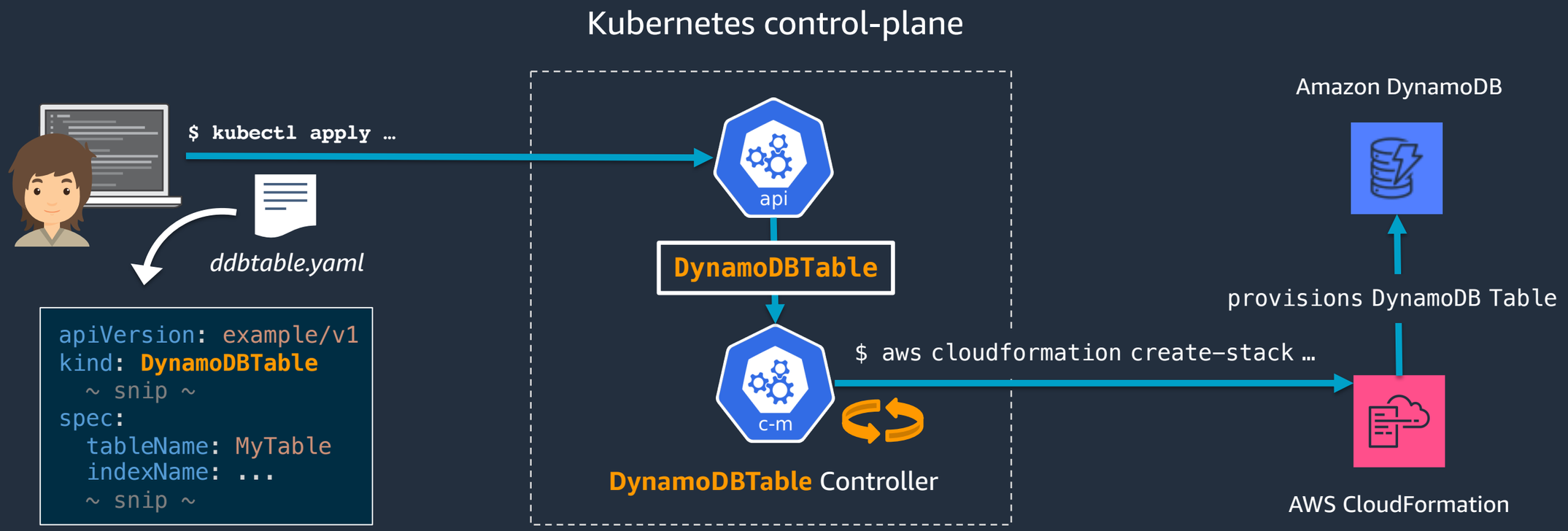
1. Aggregation Layer による Kubernetes API 拡張例

1. Metrics Server [1]
    - Horizontal/Vertical Pod Autoscaler (HPA/VPA), *kubectl top*
    - In-memory アプリケーション
2. Service Catalog [2]
    - ※ CR + Custom Controller モデルに移行中で、現行の Aggregation Layer を利用したバージョンは 2020/07 でサポート終了

[1] https://github.com/kubernetes-sigs/metrics-server
[2] https://github.com/kubernetes-sigs/service-catalog

aws

2. CR + Custom Controller による Kubernetes API 拡張例



- Custom resource そのものはいわば ConfigMap のようなもので、ビジネスロジックは持たず、Kubernetes API の永続化ストレージである etcd に保存される
- Custom resource の宣言とリソースの実体の状態差を収束させるビジネスロジックを Custom controller として実装することが一般的

aws

2. CR + Custom Controller による Kubernetes API 拡張例

i. aws/aws-app-mesh-controller-for-k8s

- AWS App Mesh のリソースを Kubernetes クラスタから管理

```yaml
apiVersion: appmesh.k8s.aws/v1beta1
kind: VirtualService
metadata:
  name: my-svc-a
  namespace: prod
spec:
  meshName: my-mesh
  virtualRouter:
    name: my-svc-a-router
    listeners:
      - portMapping:
          port: 9080
          protocol: http
  routes:
    - name: route-to-svc-a
      http:
        match:
          prefix: /
        action:
          weightedTargets:
            - virtualNodeName:
              weight: 1
```

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

aws

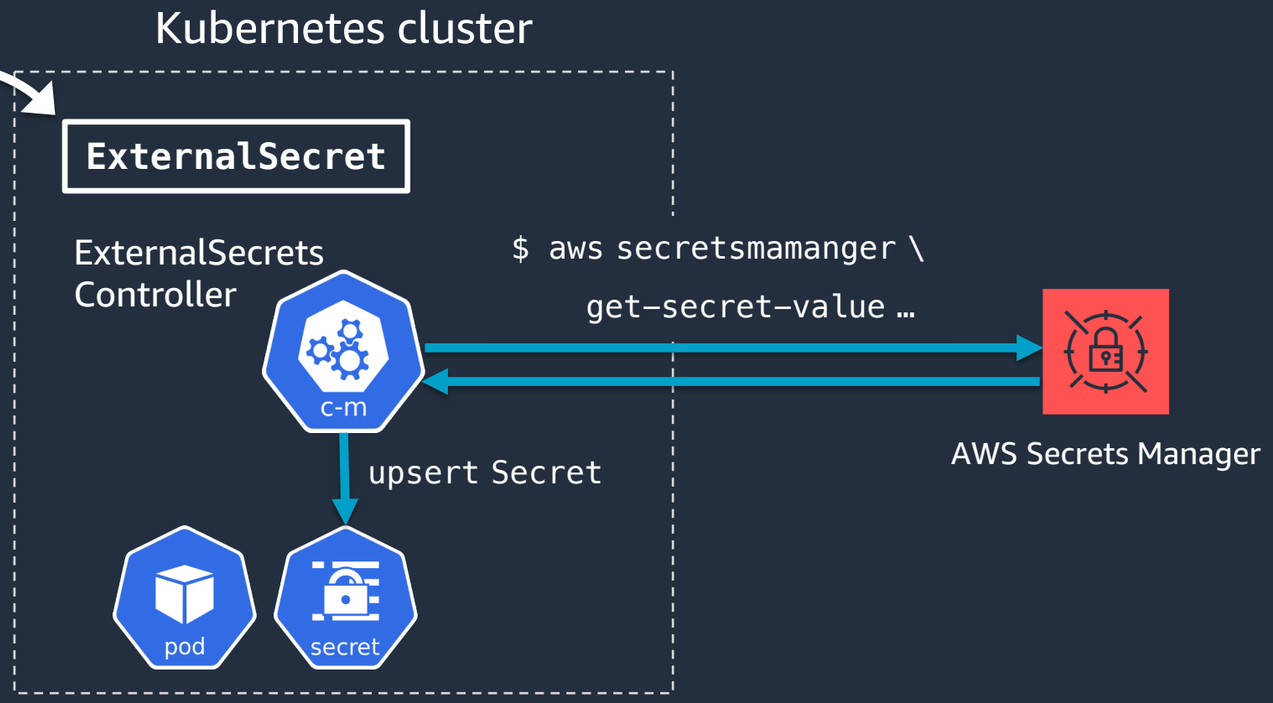
2. CR + Custom Controller による Kubernetes API 拡張例

ii. godaddy/kubernetes-external-secrets

- AWS Secrets Manager (など)のリソースを Kubernetes クラスタ内の Secrets に同期
- Pod は通常通り Secret を利用できるため透過的に AWS Secrets Manager を利用可能

my-external-secret.yaml

```
# ~ snip ~
kind: ExternalSecret
metadata:
  name: my-external-secret
spec:
  backendType: secretsManager
  data:
    key: my-service/password
    name: password
```



https://github.com/godaddy/kubernetes-external-secrets

2. CR + Custom Controller による Kubernetes API 拡張例

iii. aws/aws-service-operator-k8s [1]

- *“The ASO will allow … users to create, update, delete and retrieve the status of objects in AWS services such as S3 buckets, DynamoDB, RDS databases, SNS, etc. using the Kubernetes API …”*
- amazon-archives/aws-service-operator の v2 という立ち位置で現在開発中 [2]

iv. aws/amazon-sagemaker-operator-for-k8s [3]

- “Amazon SageMaker Operators for Kubernetes are operators that can be used to train machine learning models, optimize hyperparameters for a given model, run batch transform jobs over existing models, and set up inference endpoints.”

[1] https://github.com/aws/aws-service-operator-k8s
[2] https://github.com/aws/aws-service-operator-k8s/blob/master/docs/background.md
[3] https://github.com/aws/amazon-sagemaker-operator-for-k8s

aws

2. ”Operator” Pattern – A semantic pattern of CR + Custom controller

- CoreOS 社が提唱した Kubernetes 拡張パターン [1]
    - CR + Custom Controller による Kubernetes 拡張方法の中でも特にアプリケーション/ドメイン固有の運用知識をコード実装して自動化するパターンを “Operator” と呼ぶ (※ CR + CC 実装をすべて “Operator” と呼ぶ人々も存在する)
    - 複数の CR + Custom Controller 群によって構成されることも一般的
        - e.g. あるデータベースのセットアップ、バックアップ、リストアをそれぞれ個別の CR + Custom Controller で実装

- Operator Hub [2]
    - “*The registry for Kubernetes Operator*”
    - Launched by Red Hat “*in collaboration with AWS, Google Cloud, Microsoft*” *(Feb. 28, 2019)*

[1] https://coreos.com/blog/introducing-operators.html *(Nov. 03, 2016)*
[2] https://operatorhub.io v

aws

CR + Custom Controller による Kubernetes API の拡張

aws

Kubernetes API の拡張 – Demo

Kubernetes リソース種別

Kubernetes オブジェクト名

```yaml
apiVersion: foo.bar/v1
kind: HueLight
metadata:
  name: hallway-light
  namespace: entrance
spec:
  name: hallway
  brightness: 254
```

hue-light.yaml

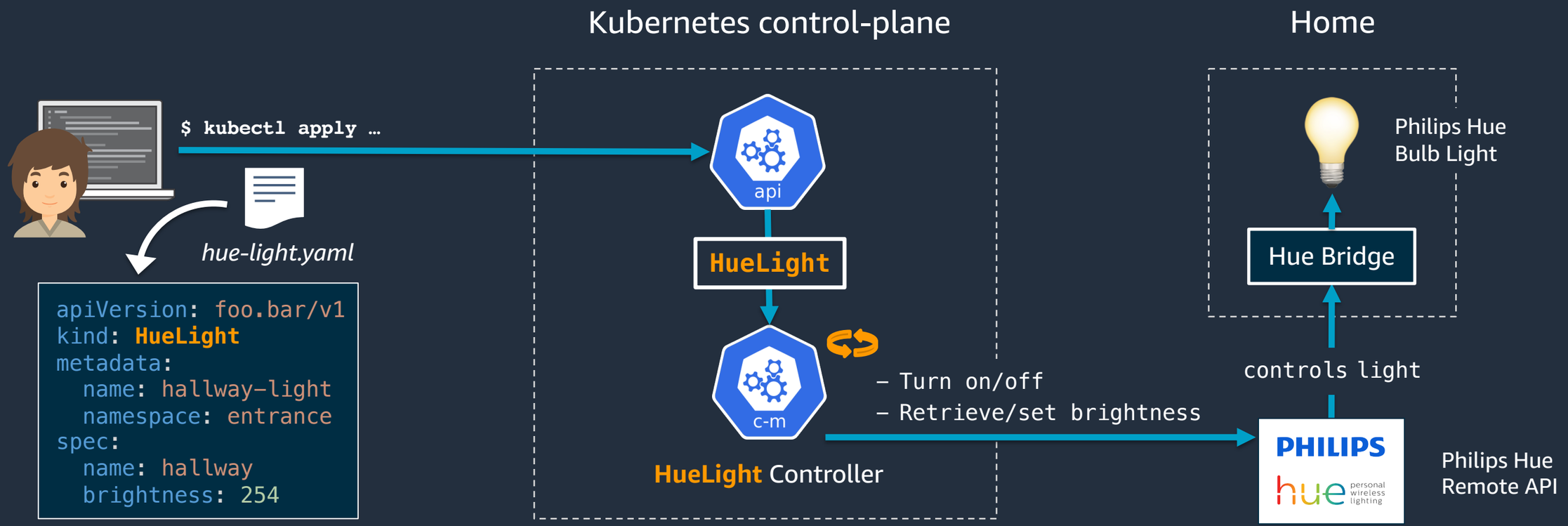
Kubernetes 名前空間
(Hue ライトが設置されている部屋の名前)

Hue ライトの名前と明るさ

Philips Hue Bulb Light

aws

Kubernetes API の拡張 – Demo



- HueLight オブジェクト作成/更新/削除 ➡ 点灯/明るさ変更/消灯
- 定期的に物理ライトの明るさを確認し、宣言と異なる場合は物理ライトの明るさを更新

aws

Demo – hue-lights-controller

当日は HueLight リソースが宣言された YAML を apply したら
廊下のライトが点灯したり明るさが変わったり消灯されたり
というオシャレなデモをやりました

aws

Kubebuilder を利用した開発をはじめる

https://github.com/kubernetes-sigs/kubebuilder

```
# プロジェクト用ディレクトリの作成
$ mkdir my-new-controller && cd $_

# ボイラープレートからプロジェクトを生成
$ kubebuilder init --domain toris.io --license apache2 --owner Tori

# ボイラープレートから API 実装の生成
$ kubebuilder create api --group home-automation --version v1 --kind HueLight

# CRD の元となる実装
$ vi api/v1/huelight_types.go

# カスタムコントローラの実装
$ vi controllers/huelight_controller.go

# Kubernetes クラスタへの CRD インストールとカスタムコントローラの実行
$ make install && make run

# あとは YAML を書いてクラスタに適用するだけ
$ cat config/samples/home-automation_v1_huelight.yaml | kubectl apply -f -
```

※ See also the Kubebuilder's Quick Start https://book.kubebuilder.io/quick-start.html

aws

Pitfalls

aws

Pitfalls

- Kubebuilder, Operator SDK [1]
    - Kubebuilder、Operator SDK が CRD + CC 実装によく利用される
    - 両者とも controller-runtime と controller-tools [2, 3] を利用しており、それぞれの最新バージョンは v0.6.0 と v0.3.0 (2020/04/30 時点)
    - 今後も破壊的変更が入る可能性がある
- Kubernetes クラスタ外リソースを触るコントローラ実装の難しさ
    - e.g. Hue ライト自体が故障していたら？誰かが電気のスイッチを切ってしまっていたら？Philips Hue Remote API が一時的に不調だったら？
    - e.g. DynamoDB Table を AWS MC で削除してしまっていたらどういう挙動を取るべき？AWS API 側のスロットリングを受けたらどうする？
    - e.g. 冪等な更新に対応していないものを扱うときは？

[1] https://github.com/operator-framework/operator-sdk
[2,3] https://github.com/kubernetes-sigs/controller-runtime, https://github.com/kubernetes-sigs/controller-tools

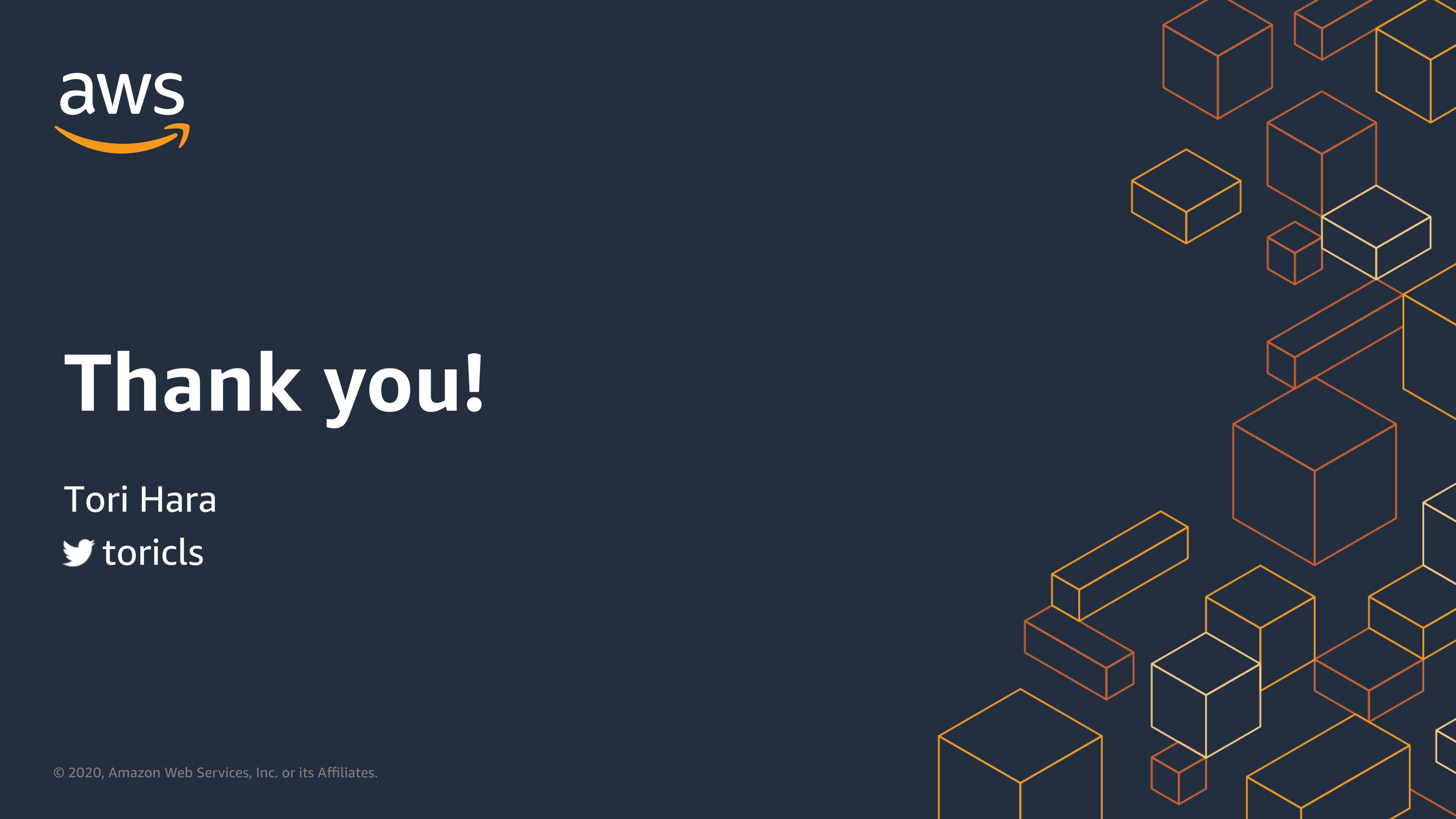
aws

まとめ

まとめ

- Kubernetes を差別化するのは高い拡張可能性と一貫性あるデプロイメントモデル
- Kubernetes API の拡張方法には Custom Resource と Aggregation Layer がある
    - 双方ともに Kubernetes 本体の実装には手を入れずにプラガブルな API 拡張が可能
    - 考慮が必要なことはまだまだたくさんある
- 要件に合わせた Kubernetes の拡張で独自の PaaS を組み上げられる
    - 開発と運用のコストが見合うのであれば

aws



Thank you!

Tori Hara

toricls