ゴール

Kubernetes環境をローカルで構築し、アプリケーションのデプロイ方法の一種である「ローリングアップデート」と「Blue/Greenデプロイ」を簡単な例で体験する。

(参考)　アプリケーションのデプロイとテストの戦略

目次

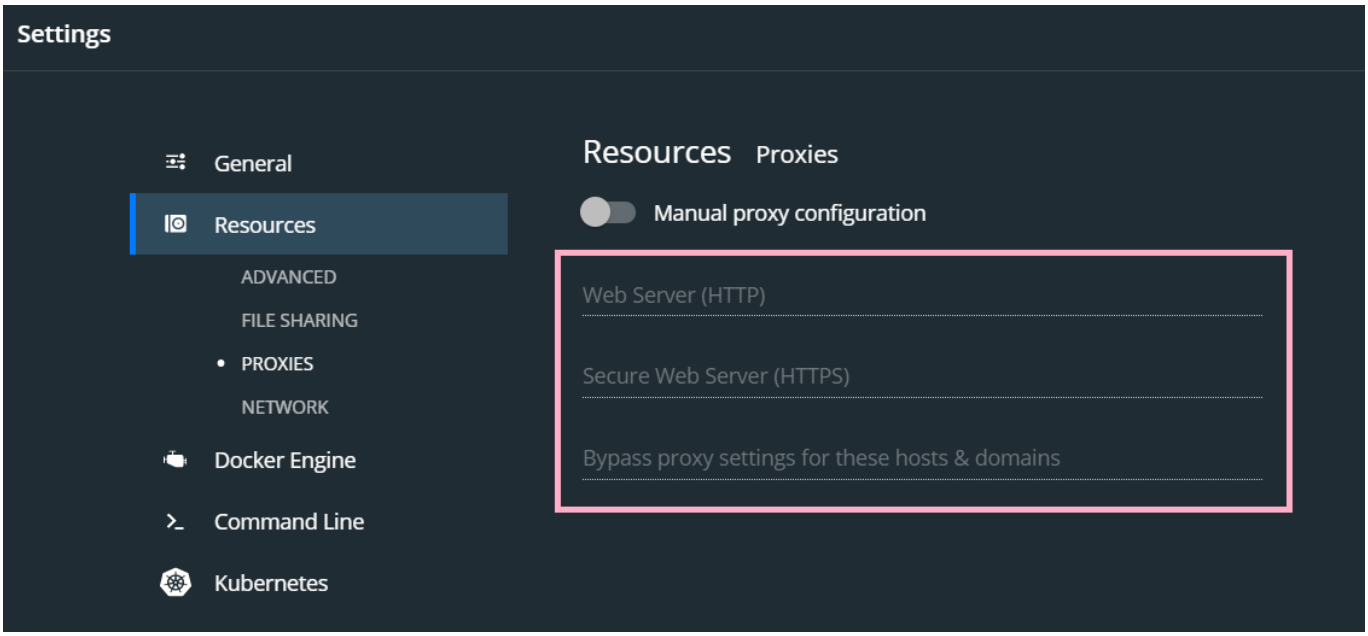
kubernetes環境構築

kubernetesのインストール

DockerDesktopからKubernetesを有効化します。

> **[重要]**
>
> 社内LAN、VPN環境ではうまく有効化できないので、DockerDesktopのPROXY設定は必ず空にすること。
> Kubernetes有効化はiphoneのテザリング環境下、または、自宅のインターネット環境下で実施します。
>
> Kubernetes関連イメージのダウンロード部分で止まり、進退窮まり初期状態にリセットせざるを得なくなります。

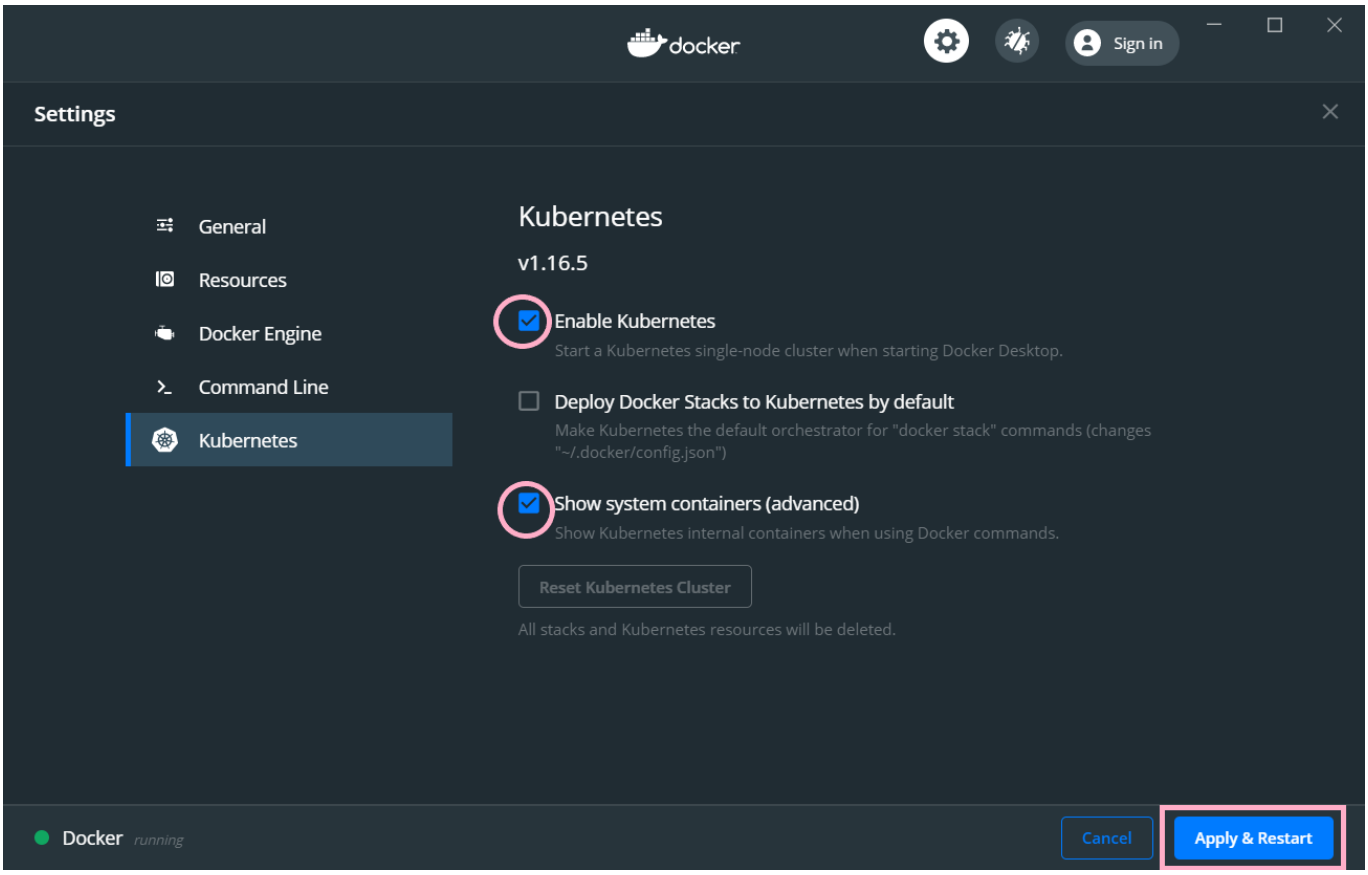
必ず PROXY設定を初期化



SettingsのResourcesでPROXIESは空白にして「Apply&Restart」ボタンを押す

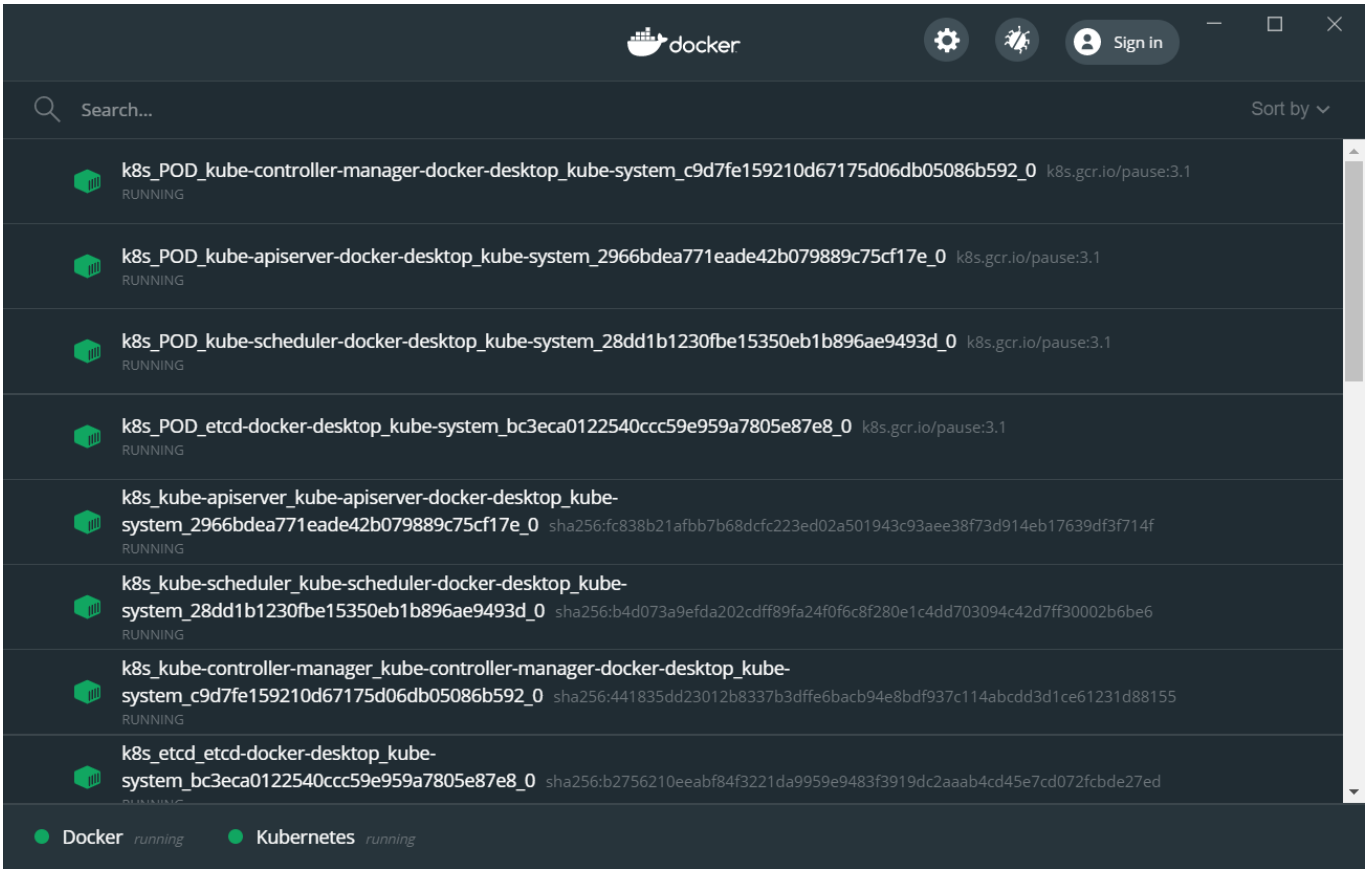
Kubernetes有効化

- Enable Kubernetesにチェックを入れる
- Show system containers(advanced)にチェックを入れる　→　インストール進捗が分かりやすい
- 「Apply&Reset」ボタンを押す



完了状態

コンテナが全てグリーンになり、画面左下のKubernetesがグリーンになりrunnningと表示されれば有効化完了です。



Kubernetes Dashboardインストール

Kubernetesの状況把握や管理のためのダッシュボードソフトをインストールします。

インストール

パワーシェルを起動し、作業フォルダにて次のコマンドを投入する。

```
> kubectl apply -f recommended.yaml
```

確認

ひきつづきパワーシェルにて起動を確認

```
> kubectl get svc -n kubernetes-dashboard
------------------------------------------------------------------------------
NAME                        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
dashboard-metrics-scraper   ClusterIP   10.111.105.244   <none>        8000/TCP
9h
kubernetes-dashboard        ClusterIP   10.108.122.103   <none>        443/TCP
9h
------------------------------------------------------------------------------
```

Dashboard管理ユーザー登録とroleのバインド

パワーシェルで次のコマンドを投入する。

```
> kubectl apply -f create_account.yaml

> kubectl apply -f role_binding.yaml
```

Ingress-nginxインストール

2日目に必要なIngress-nginxをあらかじめインストールします。

インストール

パワーシェルを起動し、作業フォルダにて次のコマンドを投入する。

```
> kubectl apply -f mandatory.yaml
------------------------------------------------------------------------------
namespace/ingress-nginx created
deployment.apps/default-http-backend created
```

```
service/default-http-backend created
configmap/nginx-configuration created
configmap/tcp-services created
configmap/udp-services created
serviceaccount/nginx-ingress-serviceaccount created
clusterrole.rbac.authorization.k8s.io/nginx-ingress-clusterrole created
role.rbac.authorization.k8s.io/nginx-ingress-role created
rolebinding.rbac.authorization.k8s.io/nginx-ingress-role-nisa-binding created
clusterrolebinding.rbac.authorization.k8s.io/nginx-ingress-clusterrole-nisa-
binding created
deployment.apps/nginx-ingress-controller created
--------------------------------------------------------------------------------

> kubectl apply -f cloud-generic.yaml
--------------------------------------------------------------------------------
service/ingress-nginx created
--------------------------------------------------------------------------------
```

確認

ひきつづきパワーシェルにて起動を確認

```
> kubectl get svc -n ingress-nginx
--------------------------------------------------------------------------------
NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
default-http-backend   ClusterIP      10.105.175.142   <none>        80/TCP
2m27s
ingress-nginx          LoadBalancer   10.105.219.79    localhost
9000:31923/TCP,443:31166/TCP   101s
--------------------------------------------------------------------------------
```

イメージファイルのビルド

予め今回使用するイメージファイルをビルド・PULLしておきます。

パワーシェルにて下記バッチを起動

```
> .\build_image.bat

--------------------------------------------------------------------------------
    :
    :
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
swtest/nginx        latest              8d71d4b00e5f        28 hours ago
170MB
```

```
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
swtest/api          v2                  51e06a4c2b89        5 hours ago
121MB
swtest/api          latest              6380831691ba        28 hours ago
121MB

REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
swtest/debug        latest              421bf1f5871d        About a minute ago
8.67MB
--------------------------------------------------------------------------------
```

Dashboard画面を起動

Dashboardにログインするための TOKENを取得

[重要] 次のコマンドは必ずパワーシェルで実行すること。

```
> kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-
dashboard get secret | sls admin-user | ForEach-Object { $_ -Split '\s+' } |
Select -First 1)

--------------------------------------------------------------------------------
Name:         admin-user-token-8db5m
Namespace:    kubernetes-dashboard
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: admin-user
              kubernetes.io/service-account.uid: 7b982cb1-97a4-4c83-ae58-
c0514d77d656

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1025 bytes
namespace:  20 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IlA1SDBab2ozYlZ4cDY4Ry1lMXE4WGc5SW5PcmgxYl9fYUVFY3NyZl
pCTVUifQ.........
--------------------------------------------------------------------------------
```

dashboardにアクセスするためのPROXYを起動する

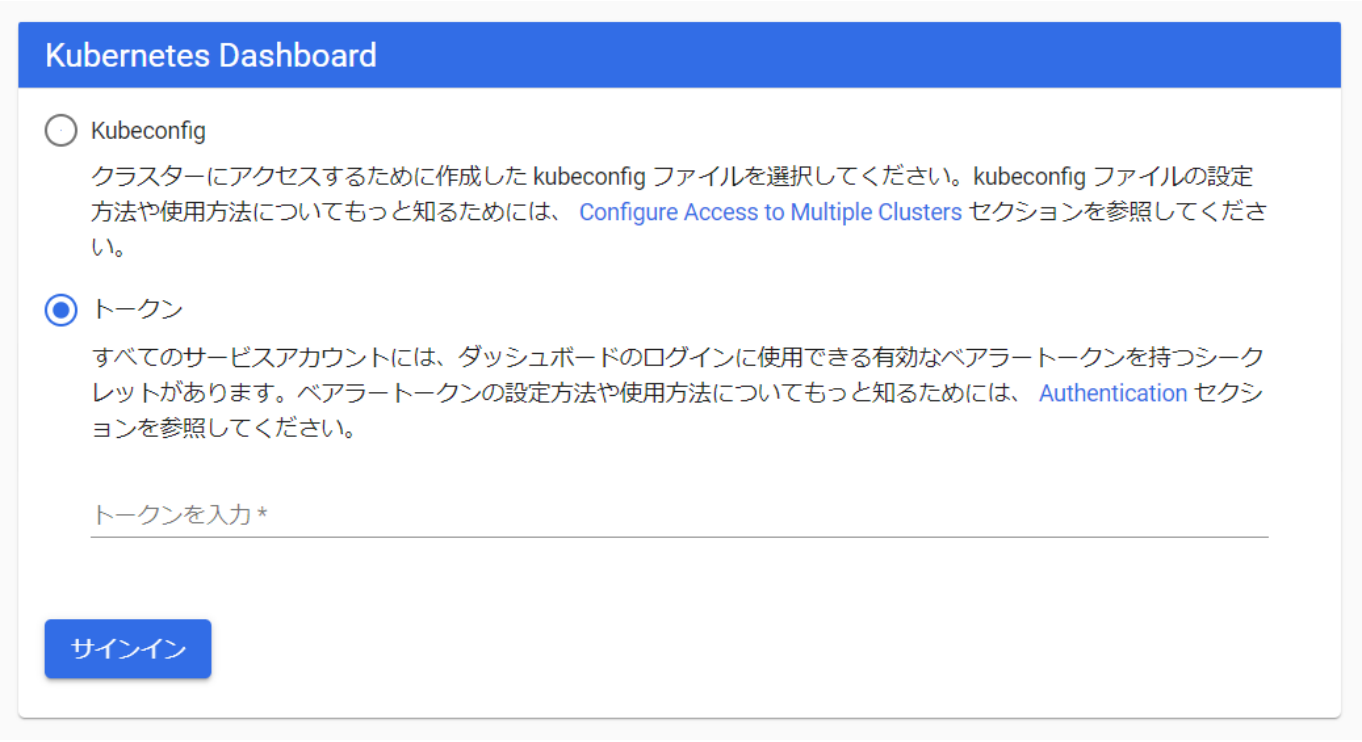
```
>kubectl proxy
```

```
----------------------------------------------------------------------------
Starting to serve on 127.0.0.1:8001
----------------------------------------------------------------------------
```

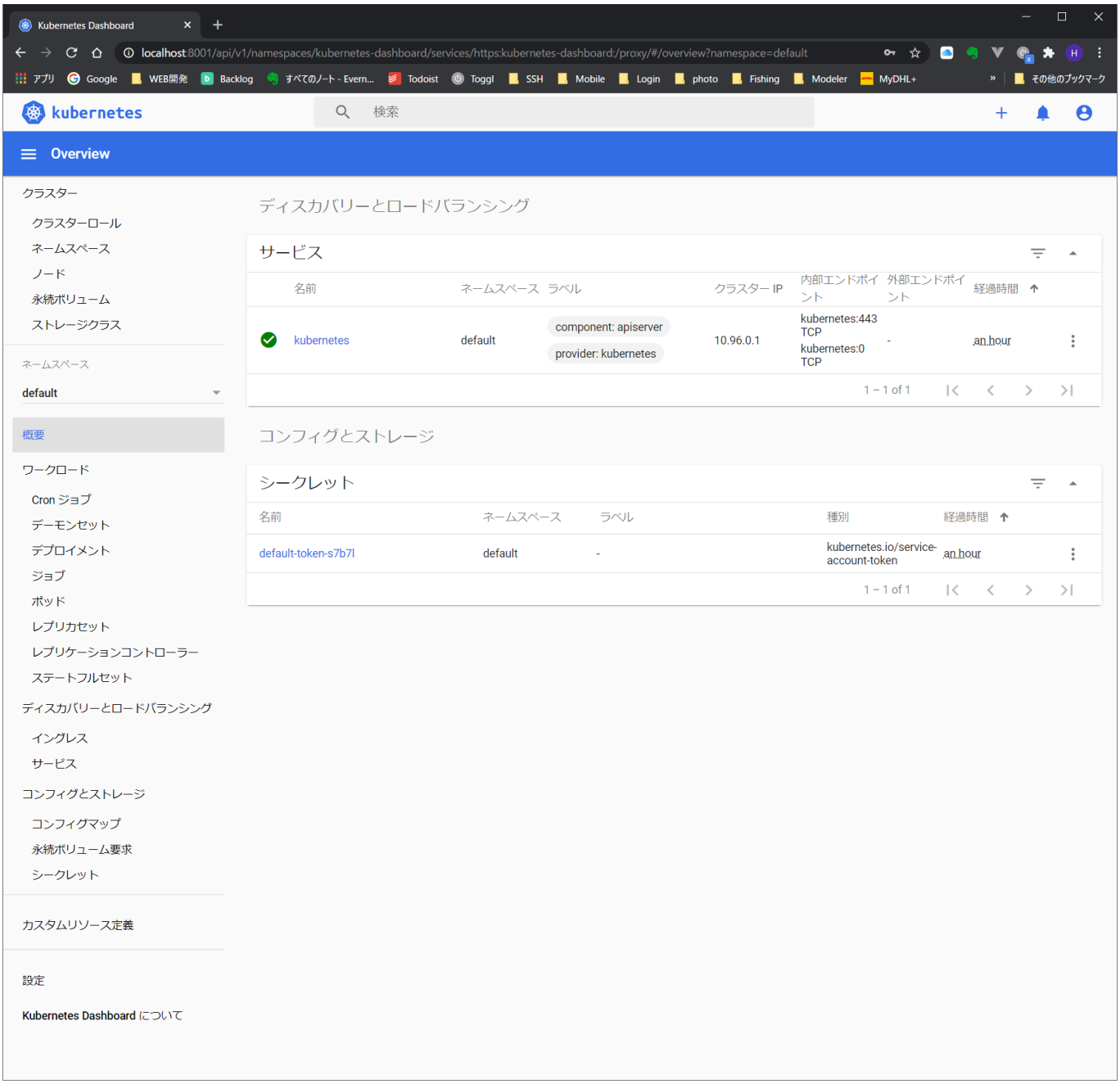
dashboard起動

下記URLをブラウザで開く

```
http://localhost:8001/api/v1/namespaces/kubernetes-
dashboard/services/https:kubernetes-dashboard:/proxy/
```



- トークンを選択する
- パワーシェル画面に表示されているTOKENをコピーし、WEB画面のトークン部分にペーストする
- 「サインイン」ボタンをクリックする



kubernetesの概念

kubernetesクラスタで実行されるアプリケーションは様々なリソースと協調して動作することで成立している。(コンテナ・オーケストレーションと呼ばれる)

以下にkubernetesのリソース一覧をまとめる。*付は今回実習で体験する。

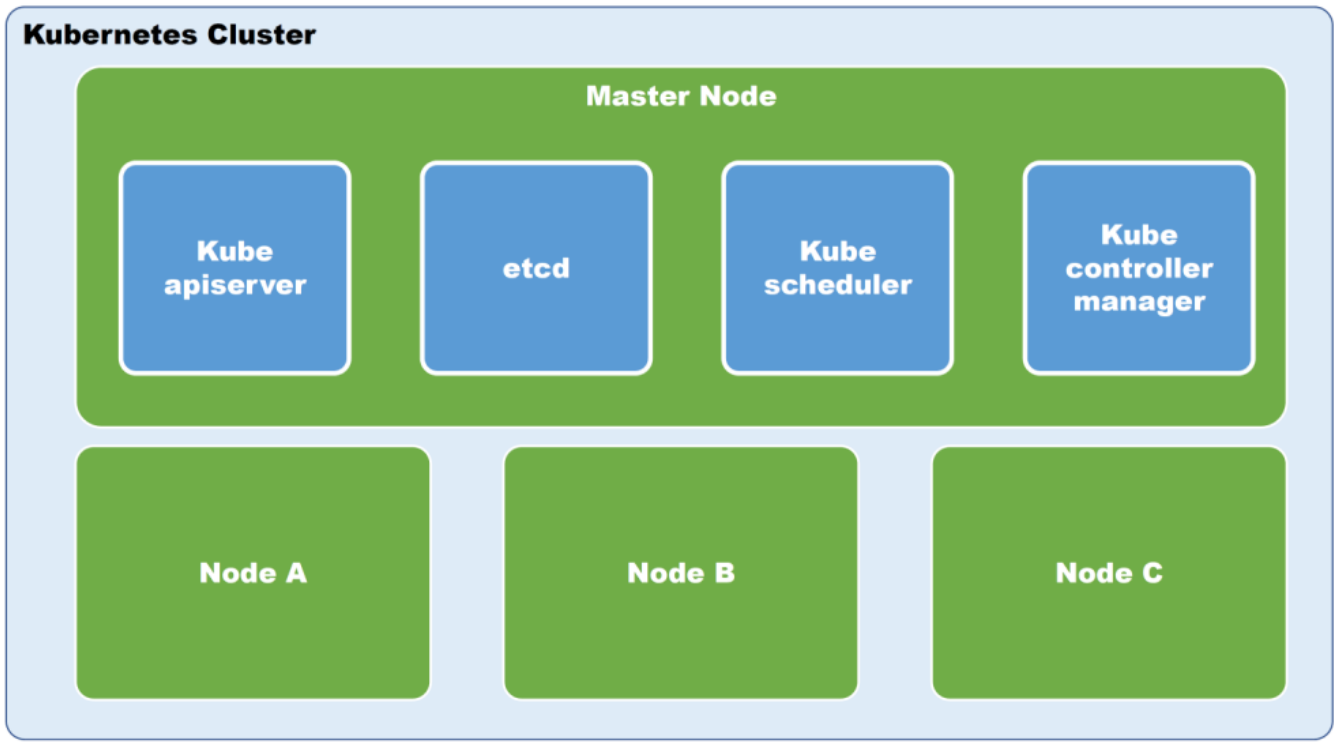
リソース名	用途
Node*	Kubernetesクラスタで実行するコンテナを配置するためのサーバー
Namespace*	Kubernetesクラスタ内で作る仮想的なクラスタ
Pod*	コンテナの集合体の単位。コンテナを実行する方法を定義する
ReplicaSet*	同じ仕様のPodを複数生成・管理する

リソース名	用途
Deployment*	ReplicaSetの世代を管理する
Service*	Podの集合にアクセスするための経路を定義する
Ingress*	ServiceをKubernetesの外部に公開する
ConfigMap	設定情報を定義し、Podに供給する
PersistentVolume	Podが利用するストレージのサイズや種別を定義する
PersistentVolumeClaim	PersisitentVolumeを動的に確保する
StorageClass	PersistentVolumeが確保するストレージの種類を定義する
StatefulSet	同じ仕様で一意性のあるPosを複数生成・管理する
Job	常駐目的ではない複数のPodを作成し、正常終了することを保証する
CronJob	cron記法でスケジューリングして実行されるJob
Secret	認証情報等の機密データを定義する
Role	Namespace内で操作可能なKubertenesリソースのルールを定義する
RoleBinding	RoleとKubernetesリソースを利用するユーザーを紐付ける
ClusterRole	Cluster全体で操作可能なKubertenesリソースのルールを定義する
ClusterRoleBinding	ClusterRoleとKubernetesリソースを利用するユーザーを紐付ける
ServiceAccont	PodにKubernetesリソースを操作させる際に利用するユーザー

kubernetesクラスタとNode

- Kubernetesクラスタは Kubernetesの様々なリソースを管理する集合体のこと。
- その中で最も大きい概念がNode（ホストのこと）となる。
- Kubernetesクラスタには全体を管理する役割のMaster Nodeが少なくとも一つは必要。

kubernetesクラスタとNode　概念図

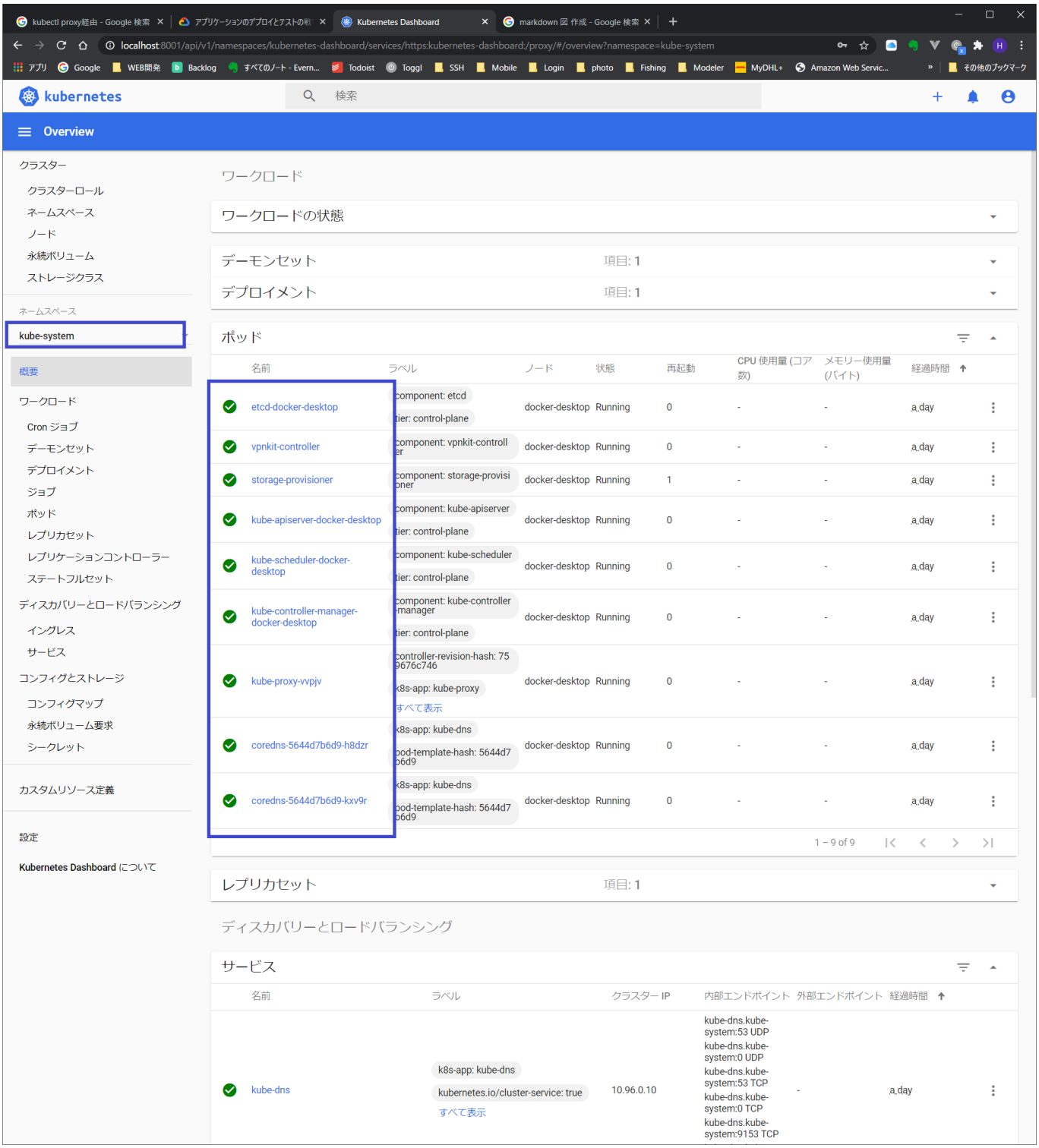


Master Nodeを構成する管理コンポーネント

コンポーネント名	役割
kube-apiserver	KubernetesのAPIを公開するコンポーネント。kubectlからのリソース操作を受け付ける
etcd	高可用性を備えた分散キーバリューストア。Kubernetesのバッキングストアとして利用される
kube-scheduler	Nodeを監視し、コンテナを配置する最適なNodeを選択する
kube-controller-manager	リソースを制御するコントローラを実行

Dashbord画面で確認

- ネームスペース：kube-system　を選択
- 概要をクリック



Namespace

NamespaceとはKubernetesクラスタ内に入れ子となる仮想的なクラスタ

ネームスペース

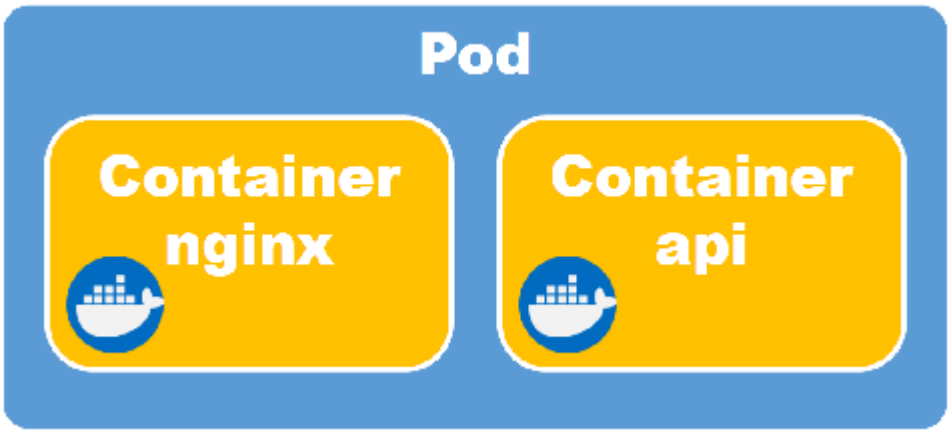
名前	ラベル	フェーズ	経過時間
ingress-nginx	-	Active	59 minutes
kubernetes-dashboard	-	Active	a day
docker	-	Active	a day
default	-	Active	a day
kube-node-lease	-	Active	a day
kube-public	-	Active	a day
kube-system	-	Active	a day

1 – 7 of 7

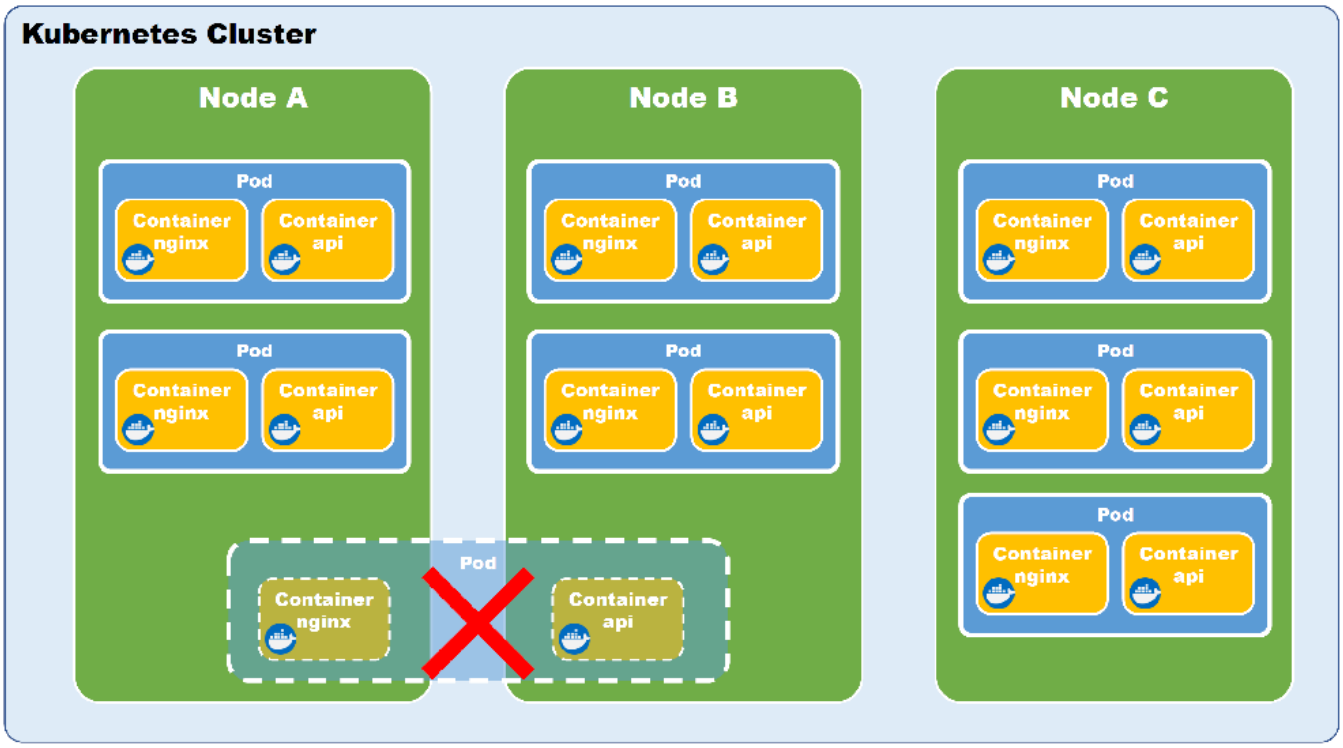
Pod

コンテナの集合体の単位で少なくともひとつのコンテナを持つ Dockerコンテナ単体、あるいはDockerコンテナの集合体

Pod単体



PodのNodeへの配置



※ひとつのPod内のコンテナが別々のノードに配置されることはない

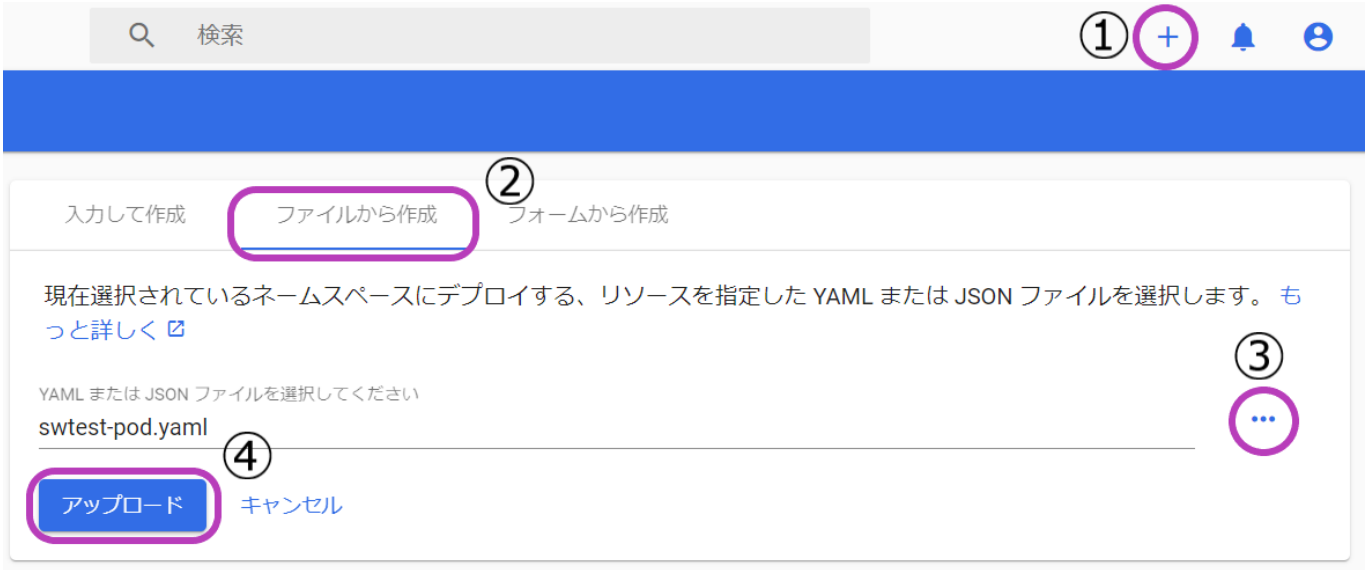
Podをデプロイ～動作確認～削除まで

マニフェストファイル: `swtest-pod.yaml`

```yaml
apiVersion: v1
kind: Pod                           # リソース種類を表す(Pod,ReplicaSet,Deployment...)
metadata:
  name: swtest
spec:
  containers:                       # Podなので配下にコンテナの仕様を記載する
  - name: nginx
    image: swtest/nginx:latest      # 使用するイメージ
    imagePullPolicy: Never          # 必ずローカルを使用
    env:
    - name: BACKEND_HOST            # 環境変数設定
      value: localhost:8080         # バックエンドのlocalhost:8080へフォワードする
    ports:
    - containerPort: 80             # コンテナポートとしてExposeする番号を指定
  - name: api
    image: swtest/api:latest        # 使用するイメージ
    imagePullPolicy: Never          # 必ずローカルを使用
    ports:
    - containerPort: 8080           # コンテナポートとしてExposeする番号を指定
```

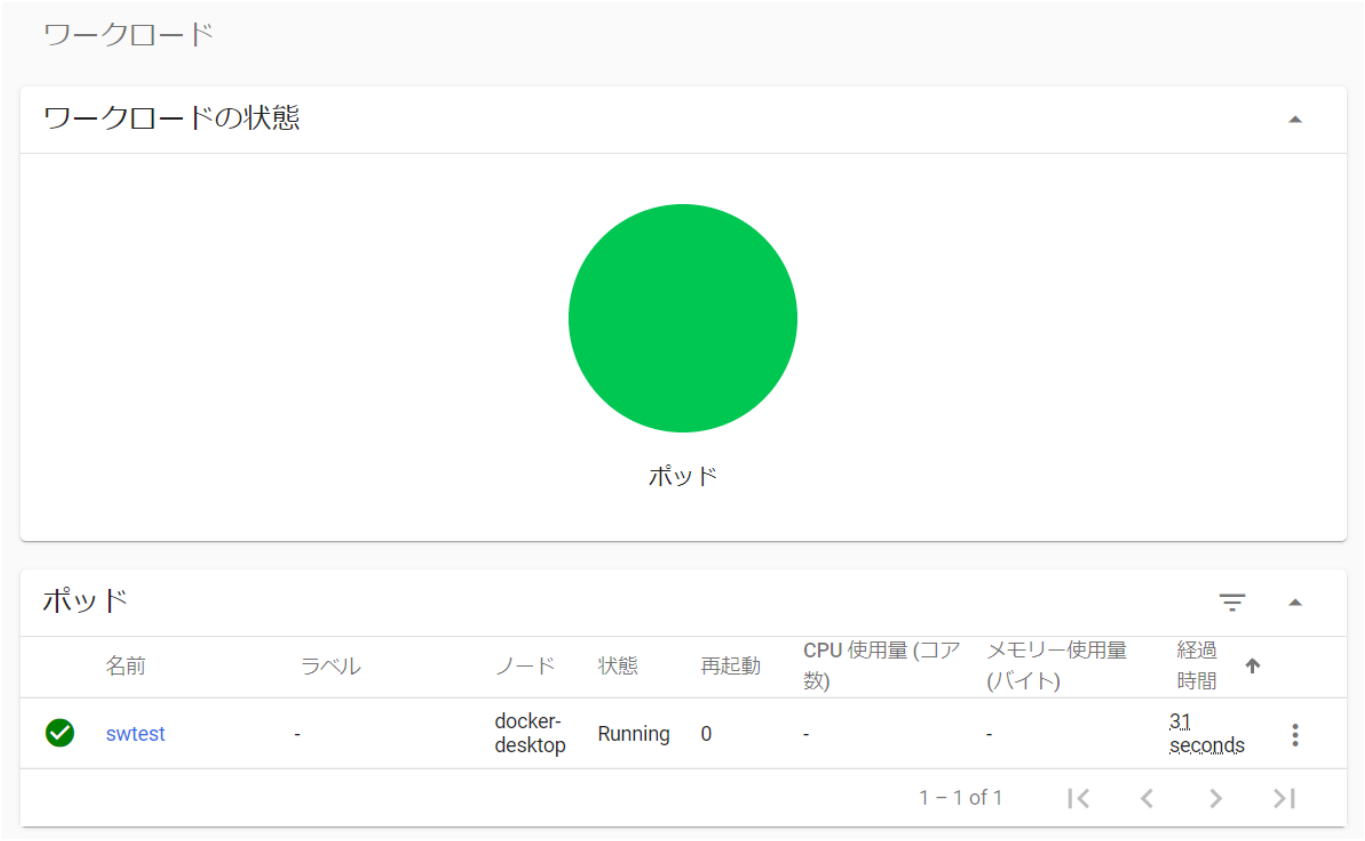
デプロイ

Dashboard画面からデプロイします



1. 画面右上「＋」マークをクリック
2. 「ファイルから作成」を選択
3. 「...」をクリックし`swtest-pod.yaml`を選択
4. 「アップロード」ボタンをクリック

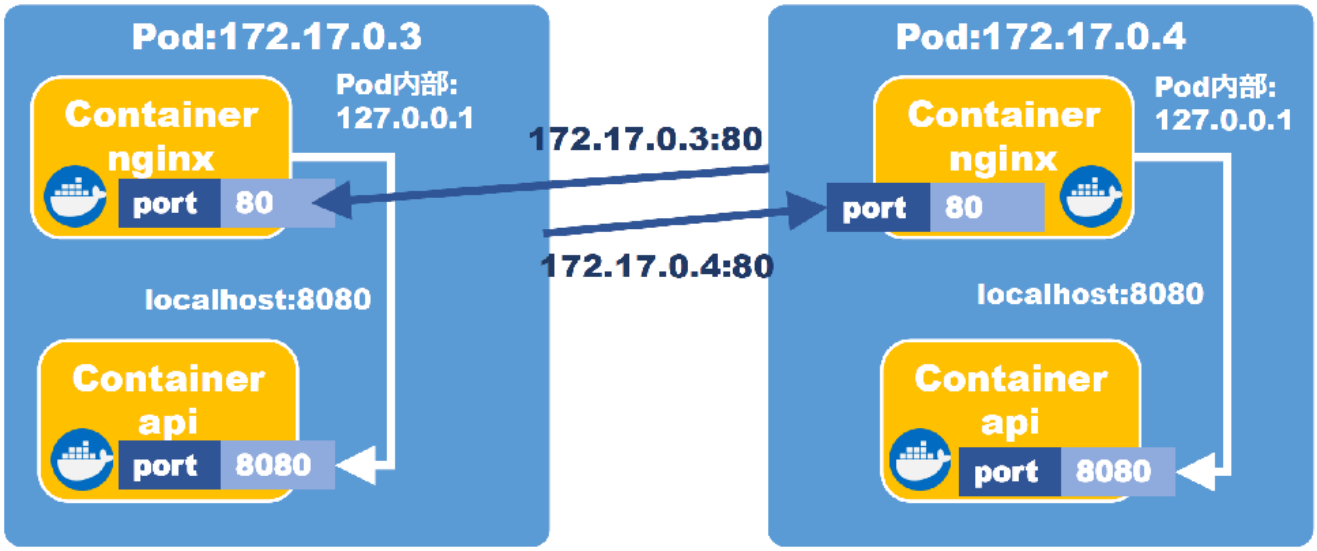
下図のようになれば正常にデプロイできています。



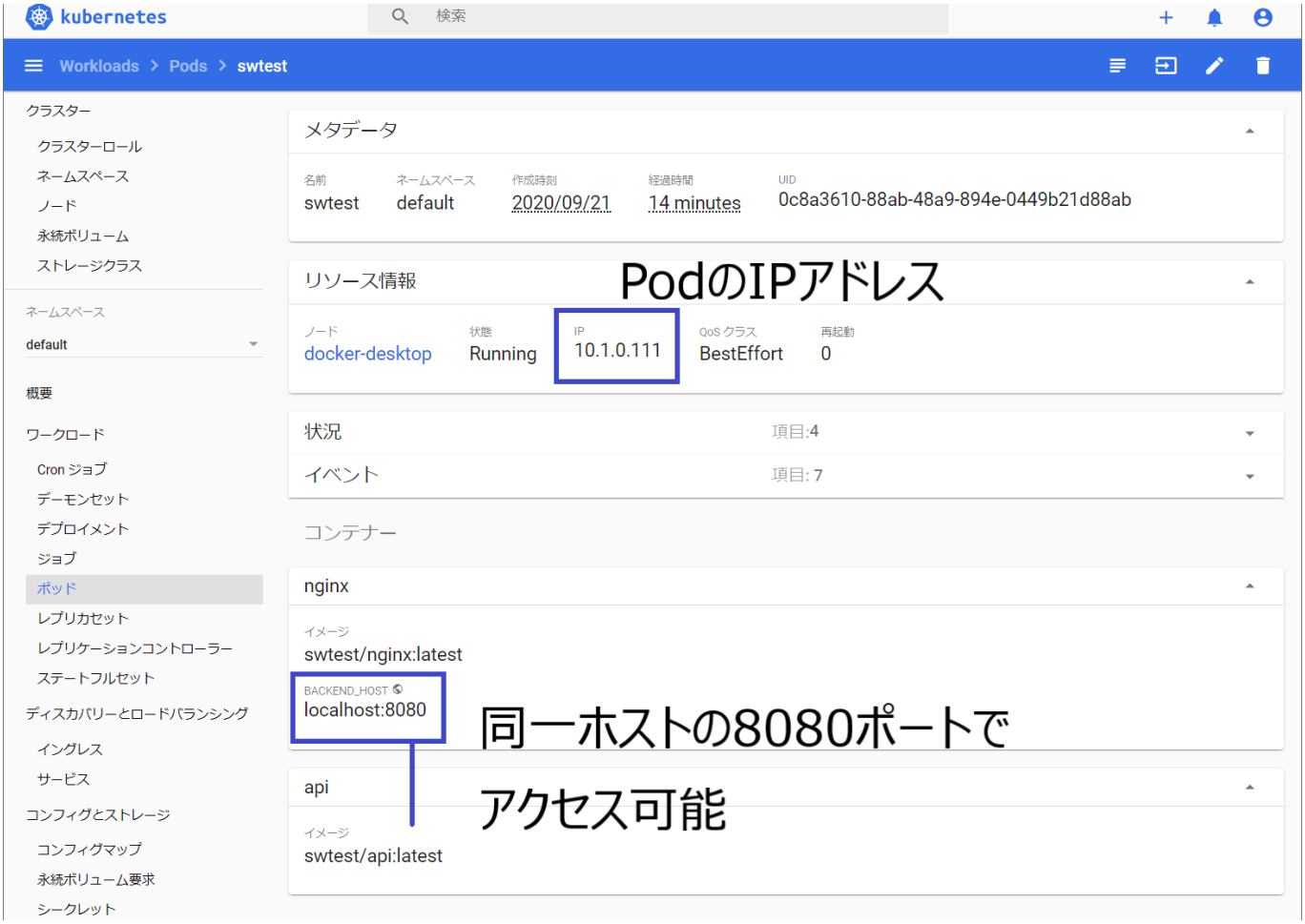
デバッグ用Podのデプロイ

`debug-pod.yaml`を同様にデプロイする

Pod仮想IPについて

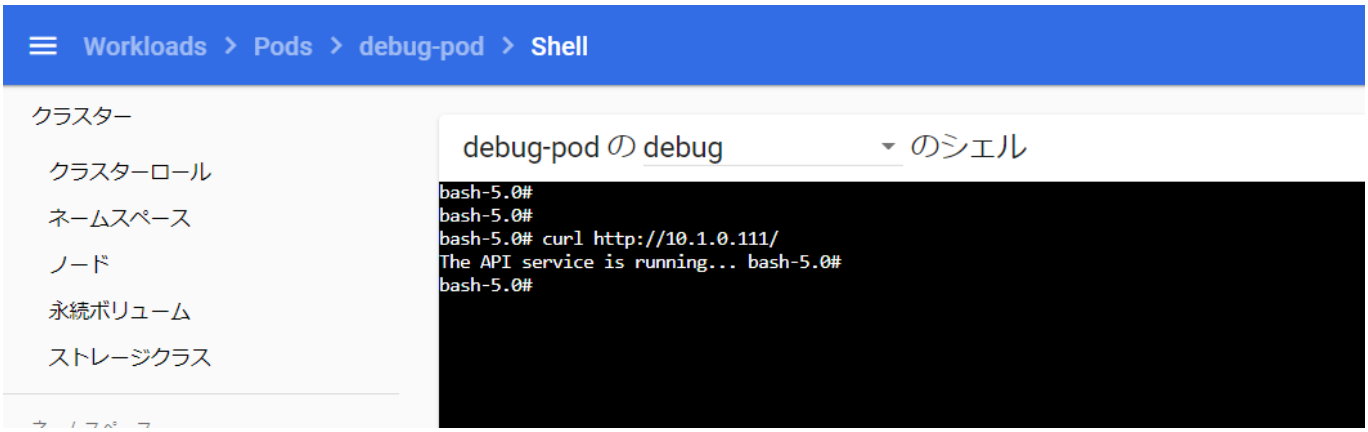


SWTESTの情報



Debugコンテナからswtestの動作確認

curlコマンドで確認



Podの削除

Dashboardを使ってswtestのPodを削除します。

メニューの「ワークロード」>「ポッド」を選択します。

ReplicaSet

同一仕様のPodの複製数を管理・制御する

Podのマニフェストファイルからはひとつのしか生成出来ない。

これを複数同時に生成可能なのが「ReplicaSet」。

マニフェストファイルの確認

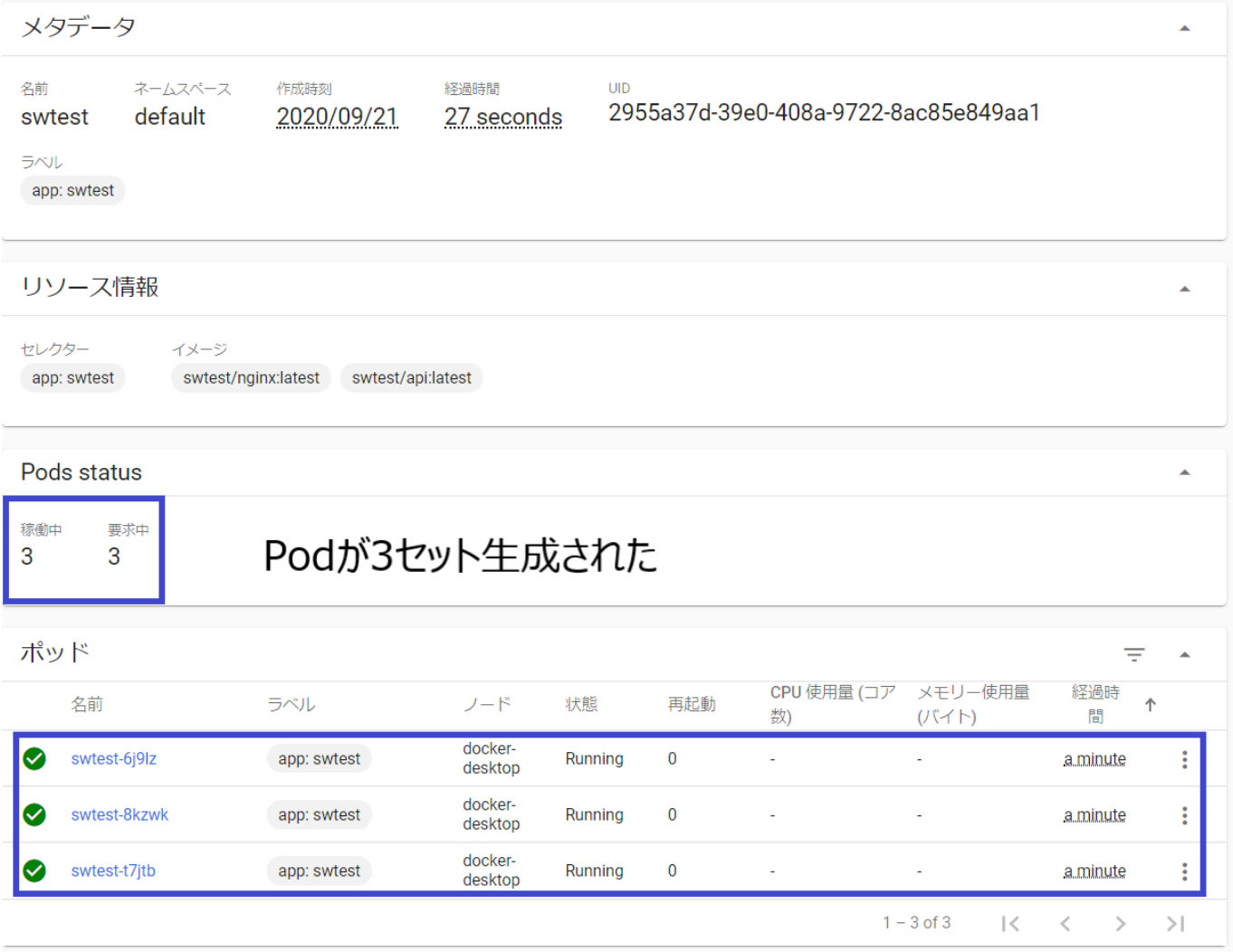
マニフェストファイル: `swtest-replicaset.yaml`

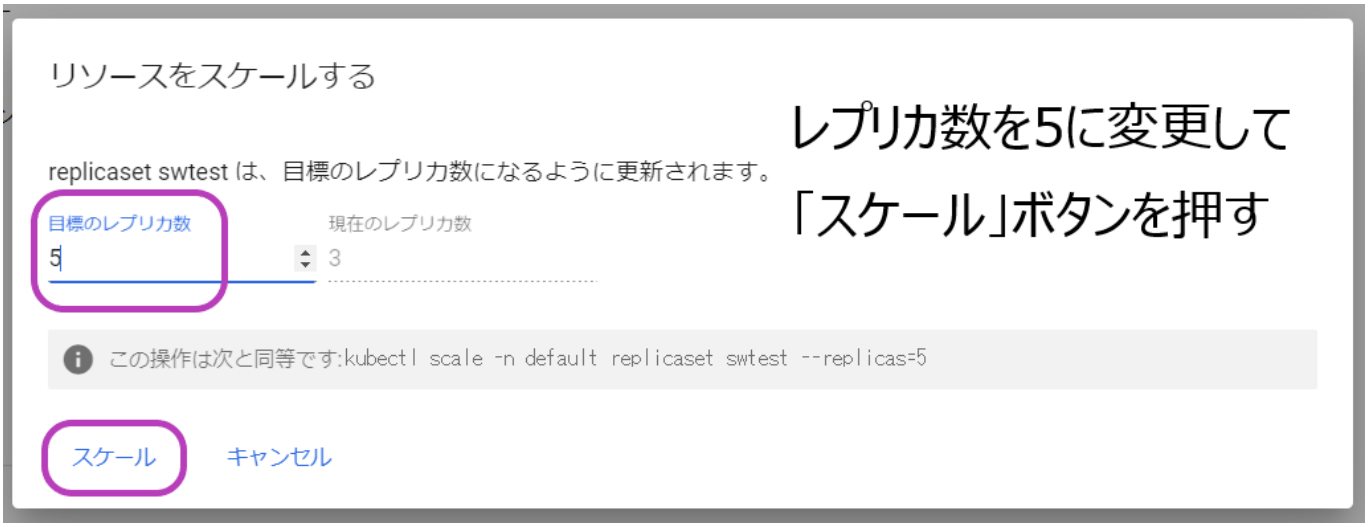
```yaml
apiVersion: apps/v1
kind: ReplicaSet            # 属性が「ReplicaSet」となる
metadata:
  name: swtest
  labels:
    app: swtest
spec:
  replicas: 3               # 作成するPodの数を指定
  selector:
    matchLabels:
      app: swtest
  template:                 # template以下はPodのそれと同様
    metadata:
      labels:
        app: swtest
    spec:
      containers:
        - name: nginx
          image: swtest/nginx:latest
          imagePullPolicy: Never
          env:
            - name: BACKEND_HOST
              value: localhost:8080
          ports:
            - containerPort: 80
        - name: api
          image: swtest/api:latest
```

```
            imagePullPolicy: Never
            ports:
              - containerPort: 8080
```

ReplicaSet生成

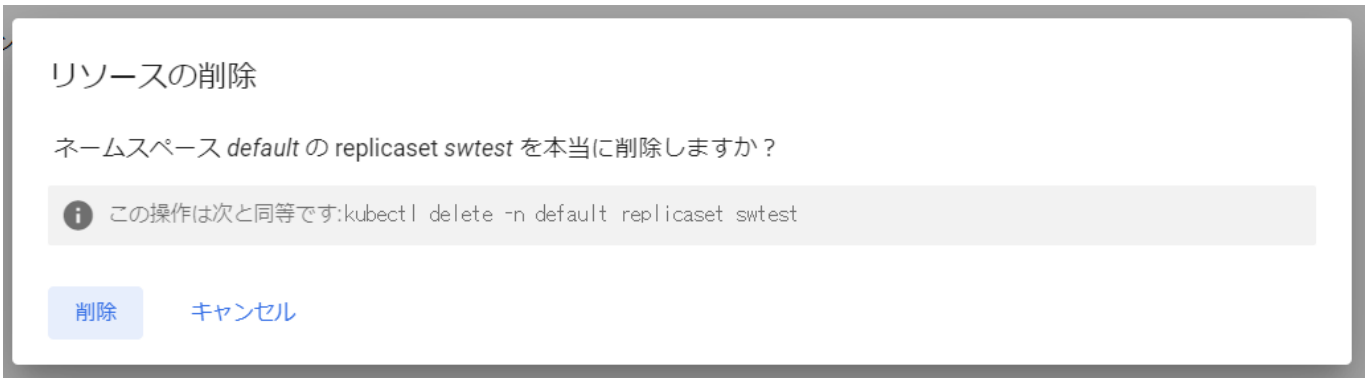
Dashboardからマニフェストファイルを選択し、ReplicaSetを生成します。





ReplicaSet削除

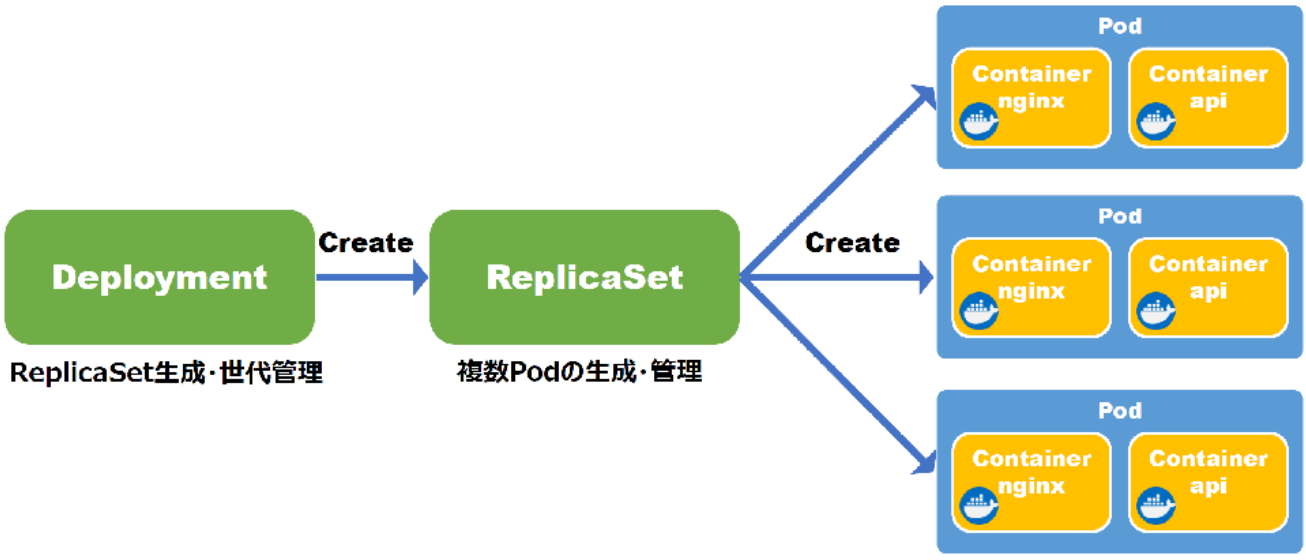
Dashboardのレプリカセットのメニューから、先ほど生成したレプリカセットを削除する。



Deployment

- **Deployment** とは、ReplicaSetの上位リソース。
- Deploymentはアプリケーションデプロイの基本単位となるリソース。
- ReplicaSetを管理・操作することができる。
- Deploymentは世代管理機能を持っておりデプロイのロールバックが可能。
- 実運用ではReplicaSetを直接用いる事は少なく、ほとんどがDeploymentのマニフェストファイルで運用している。

Deployment/ReplicaSet/Pod関連図



世代管理付のデプロイ

マニフェストファイル: `swtest-deployment.yaml`

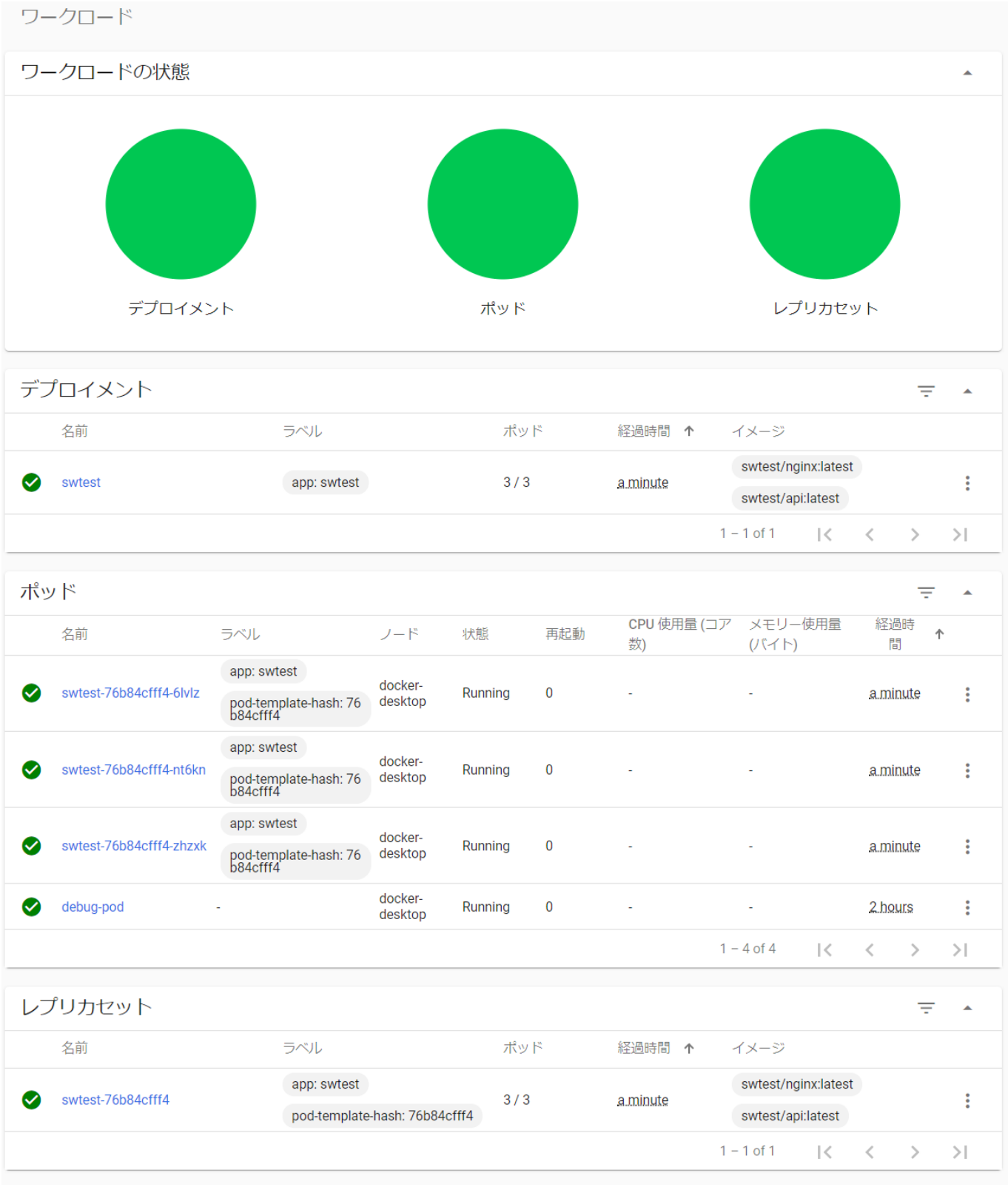
```
apiVersion: apps/v1
kind: Deployment        # ReplicaSetと異なるのはkindだけ
metadata:
```

```
    name: swtest
    labels:
      app: swtest
  spec:
    replicas: 3
    selector:
      matchLabels:
        app: swtest
    template:
      metadata:
        labels:
          app: swtest
      spec:
        containers:
          - name: nginx
            image: swtest/nginx:latest
            imagePullPolicy: Never
            env:
              - name: BACKEND_HOST
                value: localhost:8080
            ports:
              - containerPort: 80
          - name: api
            image: swtest/api:latest
            imagePullPolicy: Never
            ports:
              - containerPort: 8080
```

コマンドラインからデプロイする

作業フォルダより、下記コマンドを投入し、アプリケーションをデプロイする。

```
> kubectl apply -f swtest-deployment.yaml --record

deployment.apps/swtest created
```



リビジョンの確認

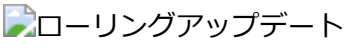
```
> kubectl rollout history deployment swtest

deployment.apps/swtest
REVISION  CHANGE-CAUSE
1         kubectl.exe apply --filename=swtest-deployment.yaml --record=true
```

ローリングアップデートとロールバック

ローリングアップデート

アプリのダウンタイムをゼロで、新しいバージョンに順次切り替えることをローリングアップデートという。

今回swtest/apiのバージョンがv2に変更になった場合を想定したローリングアップデートを行う。



マニフェストファイル

新バージョンのマニフェストファイル: `swtest-deployment-v2.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: swtest
  labels:
    app: swtest
spec:
  replicas: 3
  selector:
    matchLabels:
      app: swtest
  template:
    metadata:
      labels:
        app: swtest
    spec:
      containers:
        - name: nginx
          image: swtest/nginx:latest
          imagePullPolicy: Never
          env:
            - name: BACKEND_HOST
              value: localhost:8080
          ports:
            - containerPort: 80
        - name: api
          image: swtest/api:v2      # apiアプリバージョンがlatestからv2になった
          imagePullPolicy: Never
          ports:
            - containerPort: 8080
```

アップデートの実施

作業フォルダで下記コマンドを投入する。

```
> kubectl apply -f swtest-deployment-v2.yaml --record

deployment.apps/swtest configured
```

履歴の確認

```
> kubectl rollout history deployment swtest

deployment.apps/swtest
REVISION  CHANGE-CAUSE
1         kubectl.exe apply --filename=swtest-deployment.yaml --record=true
2         kubectl.exe apply --filename=swtest-deployment-v2.yaml --record=true
```

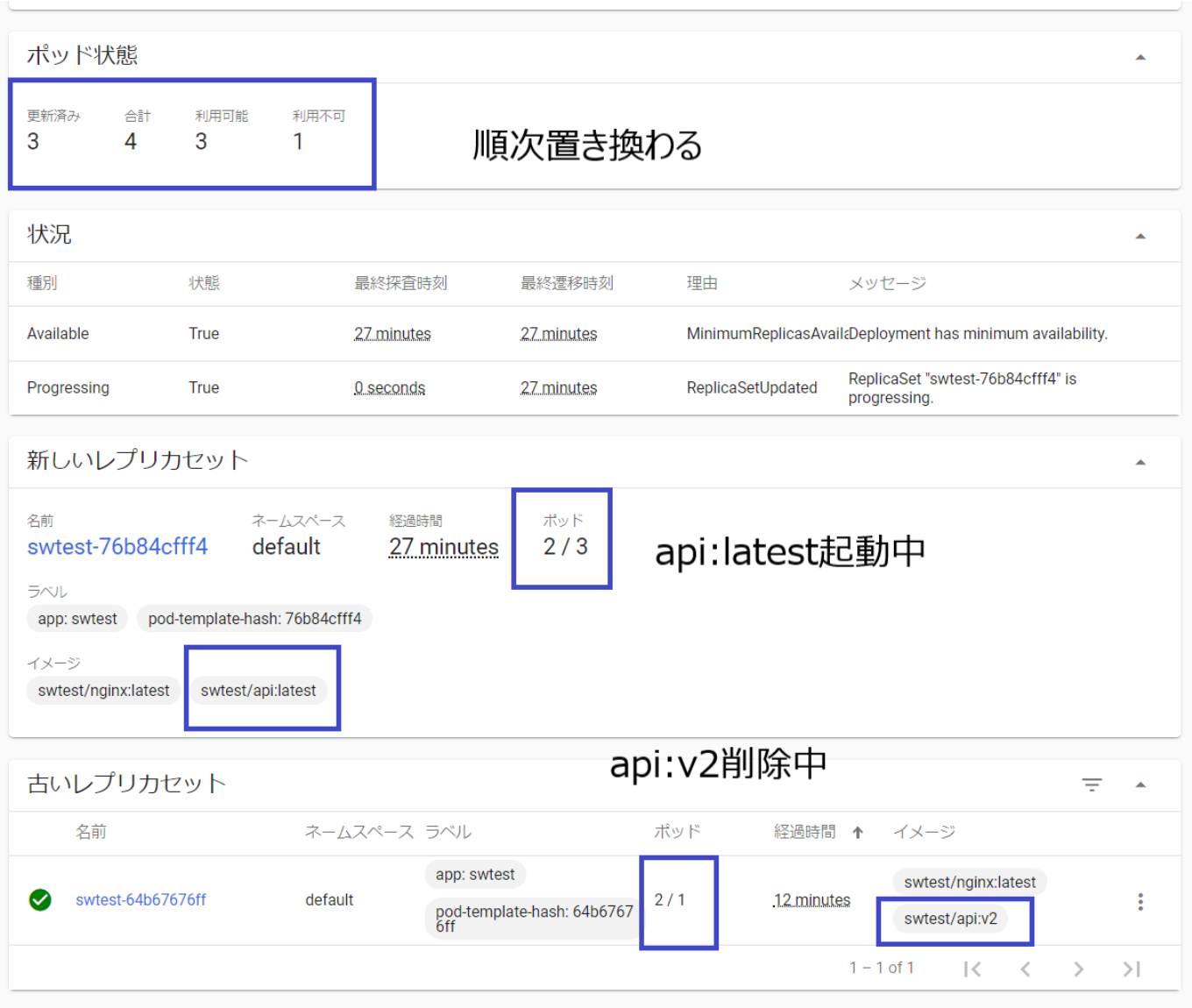
ロールバックの実施

新バージョンに何らかの問題があった場合、旧バージョンへロールバックします。

作業フォルダで下記コマンドを投入する。

```
> kubectl rollout undo deployment swtest

deployment.apps/swtest rolled back
```

切り替え中の状態



履歴の確認

```
> kubectl rollout history deployment swtest

deployment.apps/swtest
REVISION  CHANGE-CAUSE
2         kubectl.exe apply --filename=swtest-deployment-v2.yaml --record=true
3         kubectl.exe apply --filename=swtest-deployment.yaml --record=true
```

最後に削除する

```
> kubectl delete -f swtest-deployment.yaml

deployment.apps "swtest" deleted
```

Service(blue/green)作成

- **Service** とはPodの集合体(ReplicaSet)に対して経路やサービスディスカバリを提供
- Serviceのターゲットはラベルセレクタにより決定される

Blue/Greenデプロイメントを実現するため、下記2つのサービス－ReplicaSetを作成します。

1. Blueサービス → Blueデプロイメント　(swtest/api:latest)
2. Greenサービス → Greenデプロイメント　(swtest/api:v2)

blue/greenデプロイメント

マニフェストファイル: `swtest-deployment-blue.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: swtest-blue
  labels:
    app: swtest
    release: blue           # releaseというラベルでblue/greenを切り替える
spec:
  replicas: 3
  selector:
    matchLabels:
      app: swtest
      release: blue
  template:
    metadata:
      labels:
        app: swtest
        release: blue
    spec:
      containers:
        - name: nginx
          image: swtest/nginx:latest
          imagePullPolicy: Never
          env:
            - name: BACKEND_HOST
              value: localhost:8080
          ports:
            - containerPort: 80
        - name: api
          image: swtest/api:latest
```

```
          imagePullPolicy: Never
          ports:
            - containerPort: 8080
```

マニフェストファイル: swtest-deployment-green.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: swtest-green
  labels:
    app: swtest
    release: green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: swtest
      release: green
  template:
    metadata:
      labels:
        app: swtest
        release: green
    spec:
      containers:
        - name: nginx
          image: swtest/nginx:latest
          imagePullPolicy: Never
          env:
            - name: BACKEND_HOST
              value: localhost:8080
          ports:
            - containerPort: 80
        - name: api
          image: swtest/api:v2          # 新バージョンのAPI
          imagePullPolicy: Never
          ports:
            - containerPort: 8080
```

blue/greenサービス

マニフェストファイル: swtest-servvice.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: swtest-blue
spec:
```

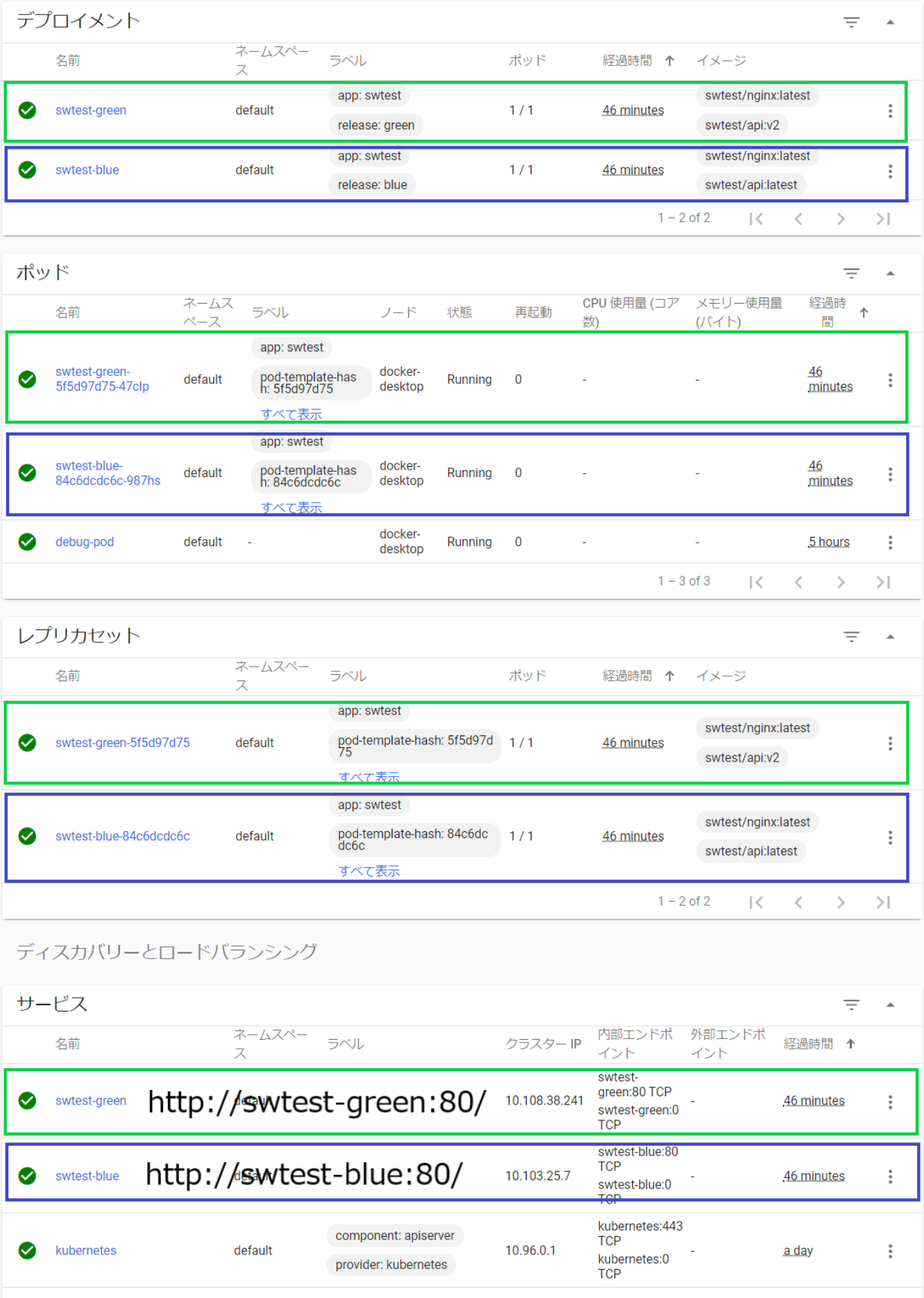
```
  selector:
    app: swtest
    release: blue       # release=blueのReplicaSetはswtest-blueサービス
  ports:
    - name: http        # httpの80番ポートをswtestに振り当てる
      port: 80

---
apiVersion: v1
kind: Service
metadata:
  name: swtest-green
spec:
  selector:
    app: swtest
    release: green      # release=greenのReplicaSetはswtest-greenサービス
  ports:
    - name: http        # httpの80番ポートをswtestに振り当てる
      port: 80
```

デプロイ

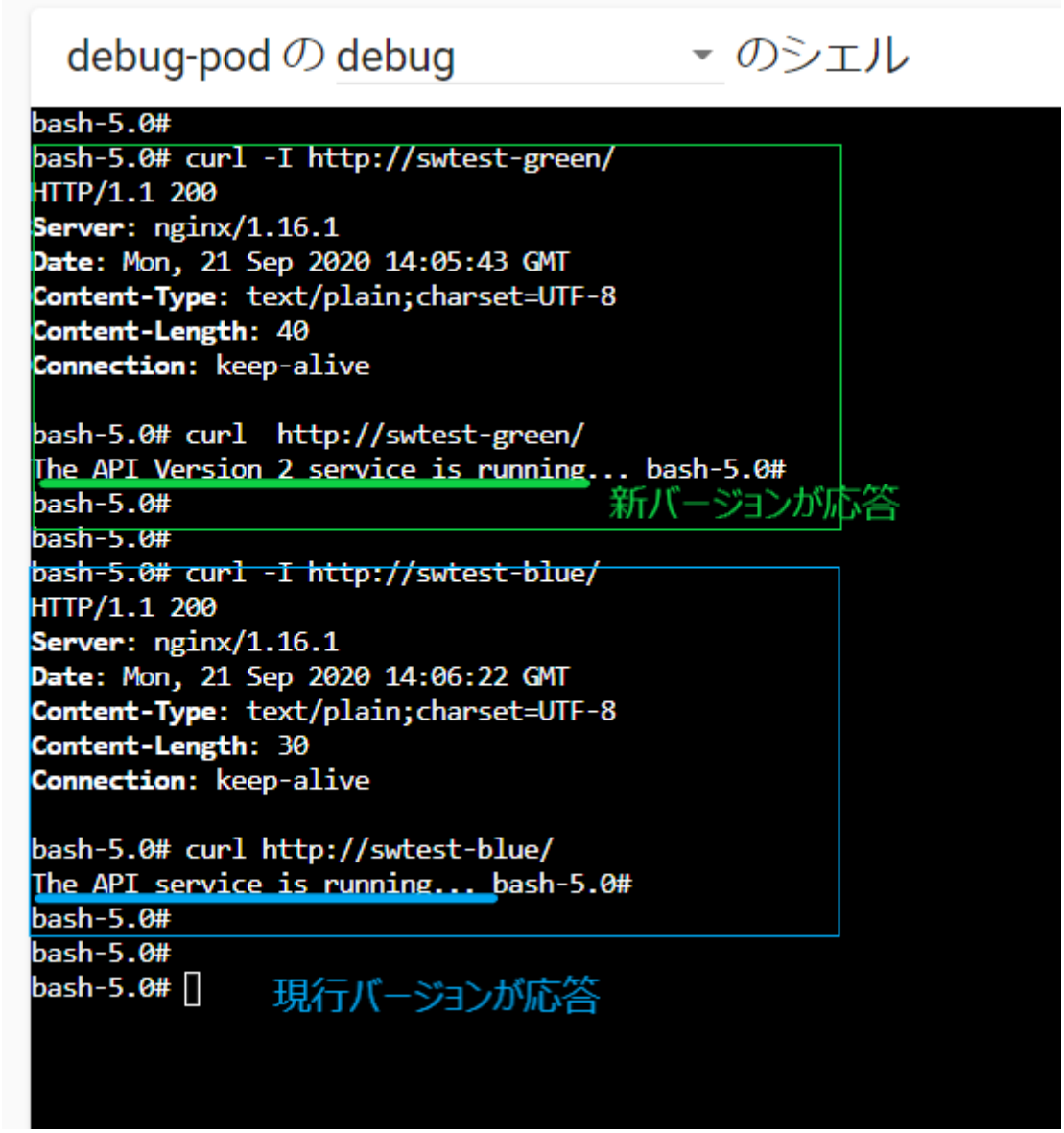
作業フォルダで下記コマンドを投入する。

```
> kubectl apply -f swtest-deployment-blue.yaml --record

> kubectl apply -f swtest-deployment-green.yaml --record

> kubectl apply -f swtest-service.yaml
```

状態確認

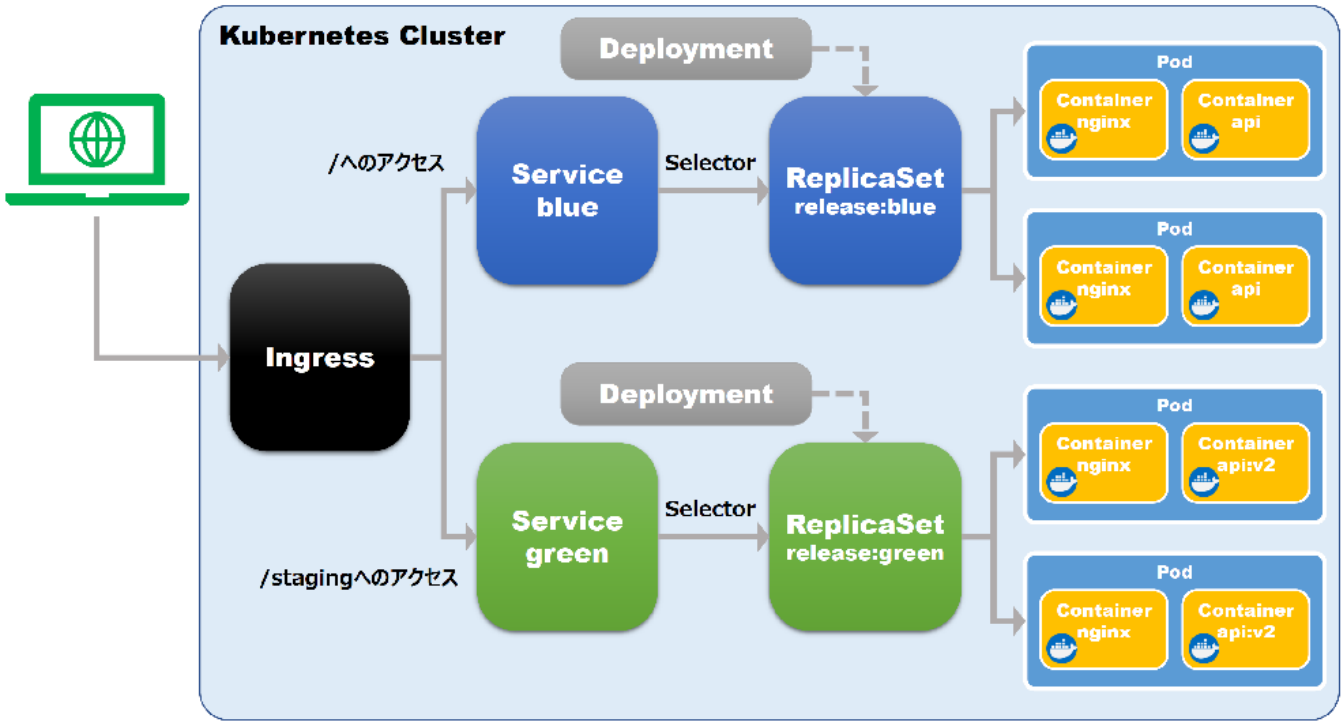


debugコンテナからアクセスし動作確認



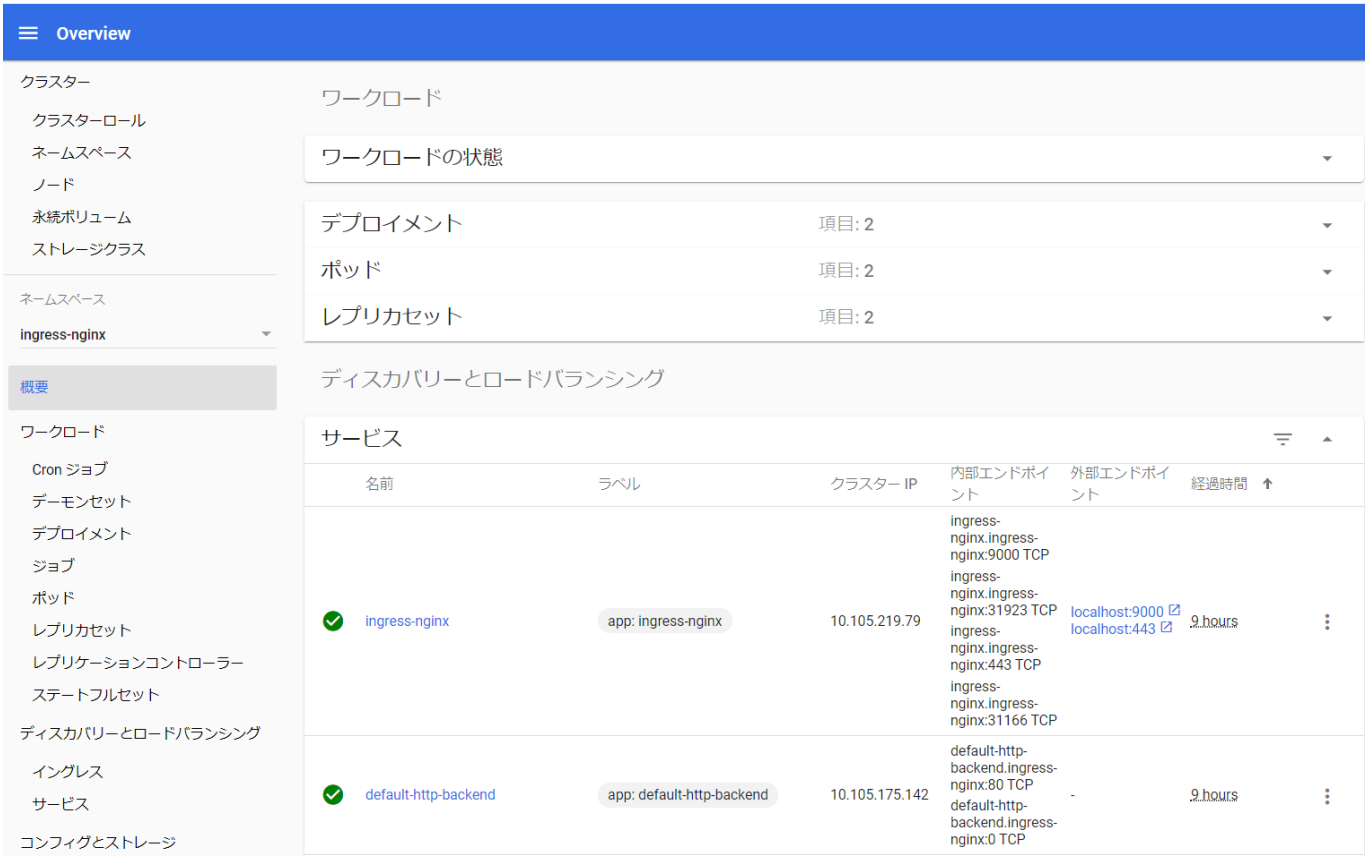
IngressによるBlue/Greenデプロイ

IngressによるBlue/Greenデプロイイメージ

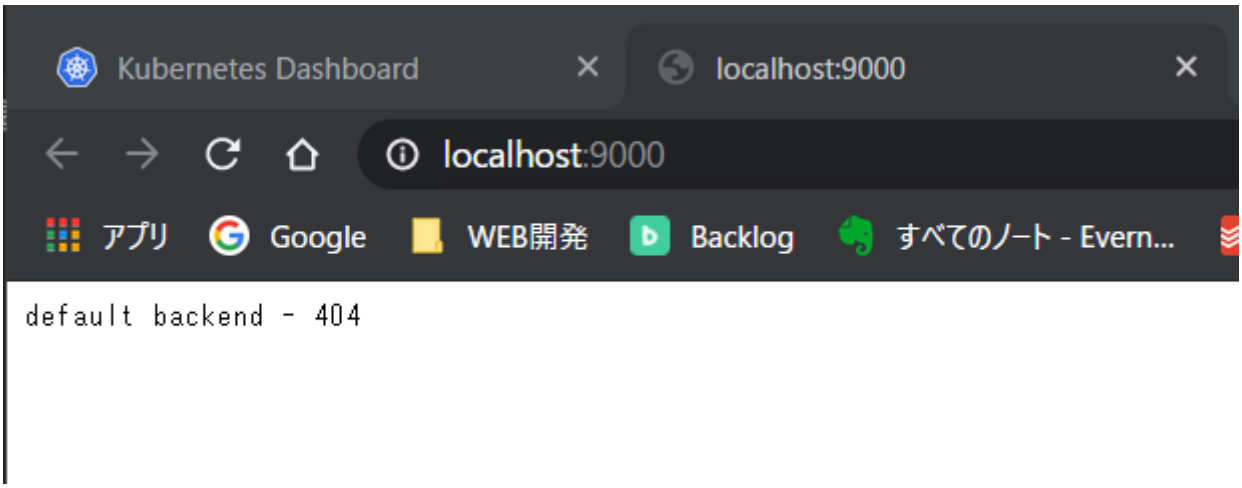


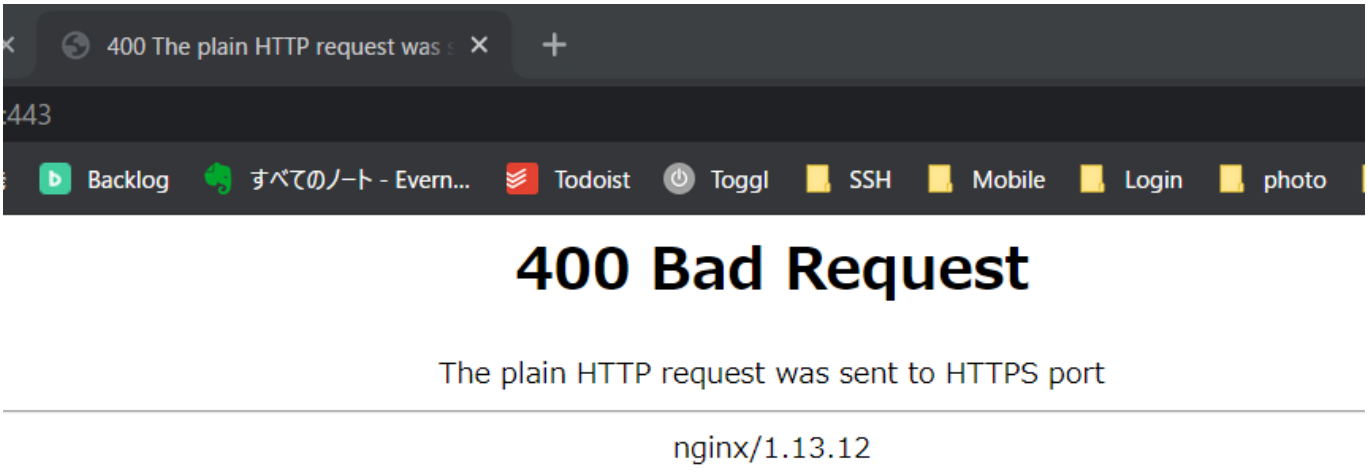
Ingress-nginxの状況確認

メニューのネームスペースから `ingress-nginx` を選択し、メニューの「概要」をクリックすると下記画面表示となる。



画面中の「外部エンドポイント」のURLリンクをクリックすると、下記画面が新たに開けば正常に動作している。





Ingress追加

マニフェストファイル: `swtest-ingress.yaml`

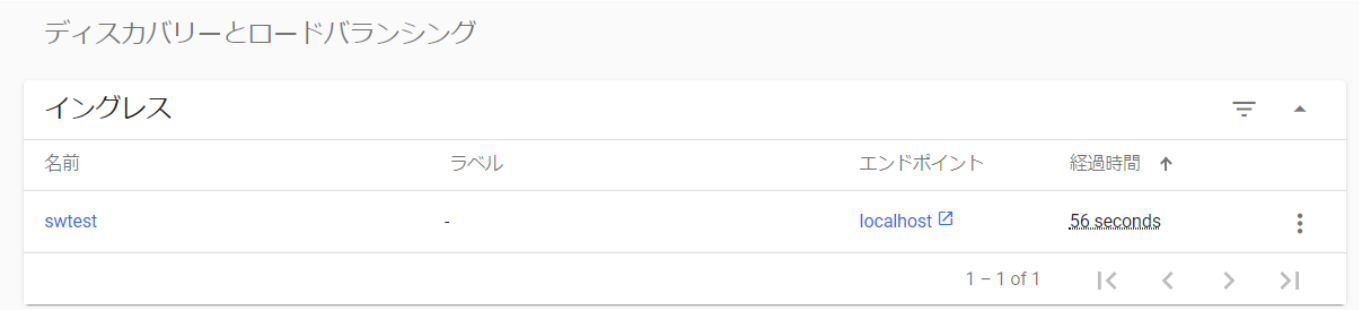
```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: swtest
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: swtest.api.local
      http:
        paths:
          - path: /                          # 現行バージョンは/でアクセス
            backend:
              serviceName: swtest-blue
```

```
            servicePort: 80
        - path: /staging                    # 新バージョンは/stagingでアクセス
          backend:
            serviceName: swtest-green
            servicePort: 80
```

作業フォルダで下記コマンドを投入する

```
> kubectl apply -f swtest-ingress.yaml

ingress.networking.k8s.io/swtest configured
```

下図のようにイングレスが追加される



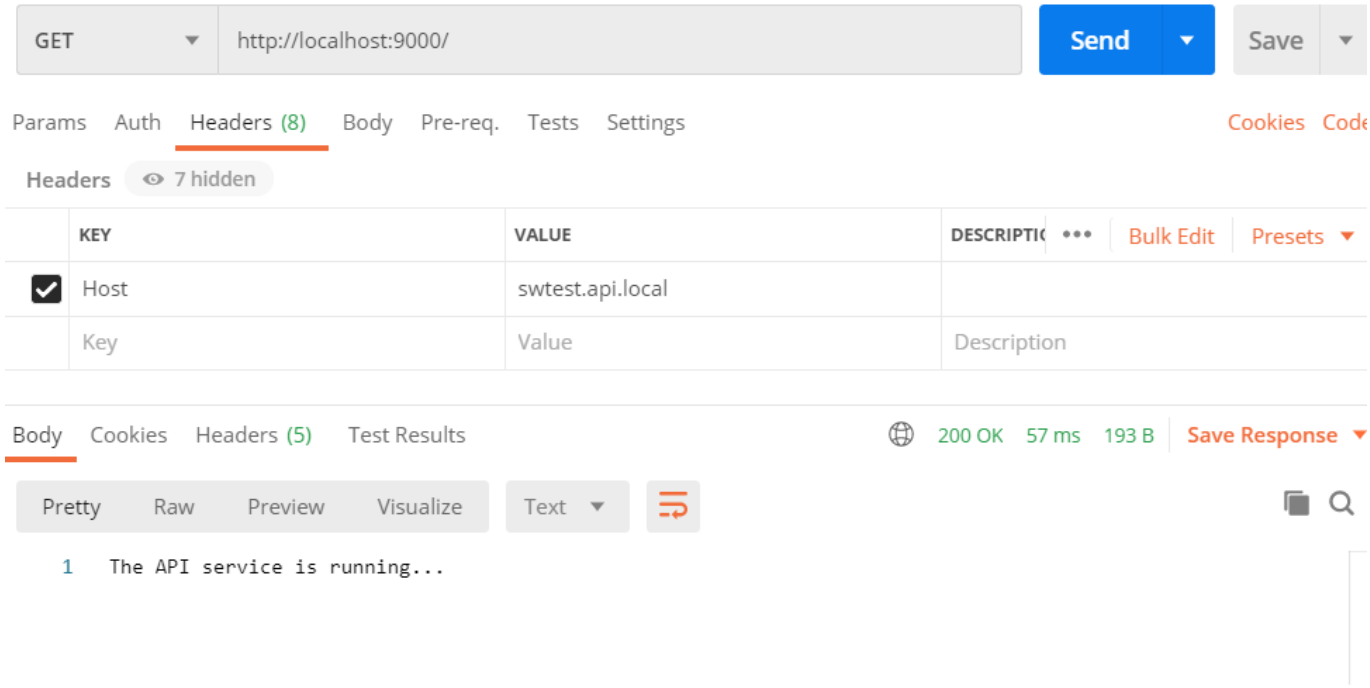
動作確認

現行バージョンへのアクセス

Postmanで動作確認を行います

- URL: http://localhost:9000/
- Headers: Host -> swtest.api.local
- 「Send」ボタンクリック

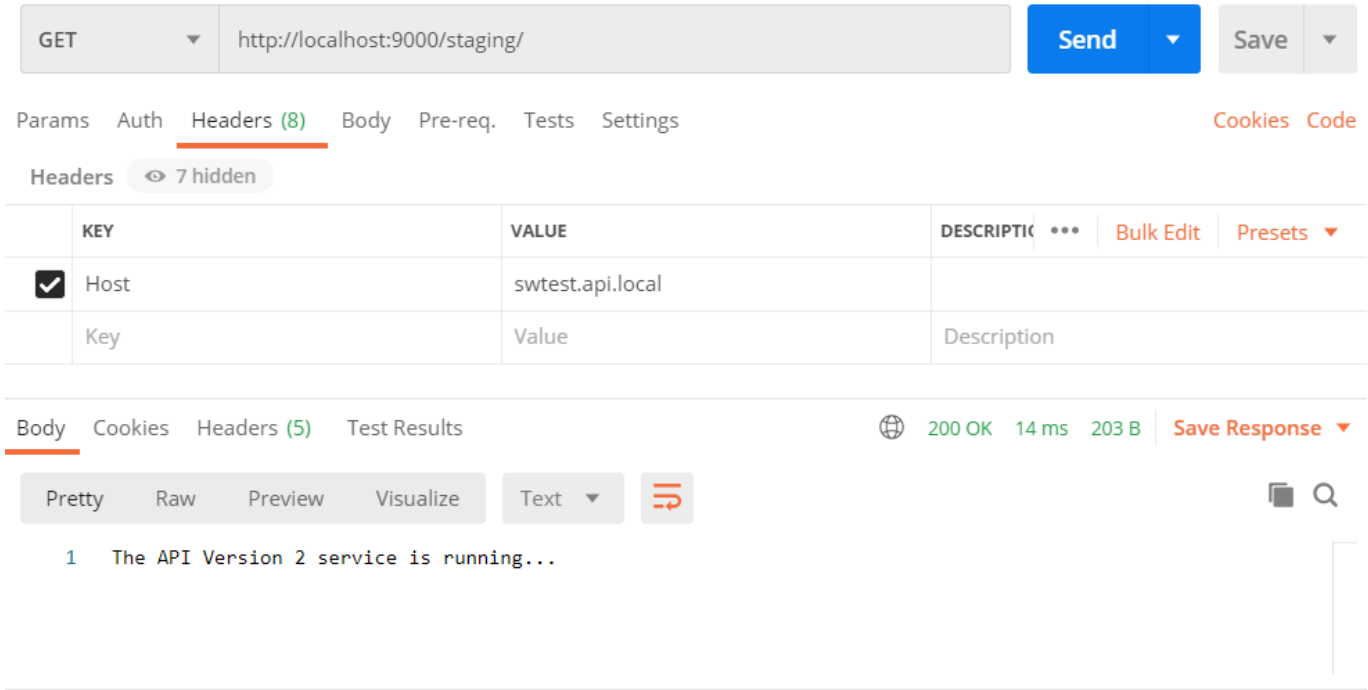
現行バージョンの応答が返る



次期バージョンへのアクセス

- URL: http://localhost:9000/staging/
- Headers: Host -> swtest.api.local
- 「Send」ボタンクリック

次期バージョンの応答が返る



Blue -> Green への切り替え

path切り替え

マニフェストファイルの「path」を入れ替えることによりGreenをリリースする

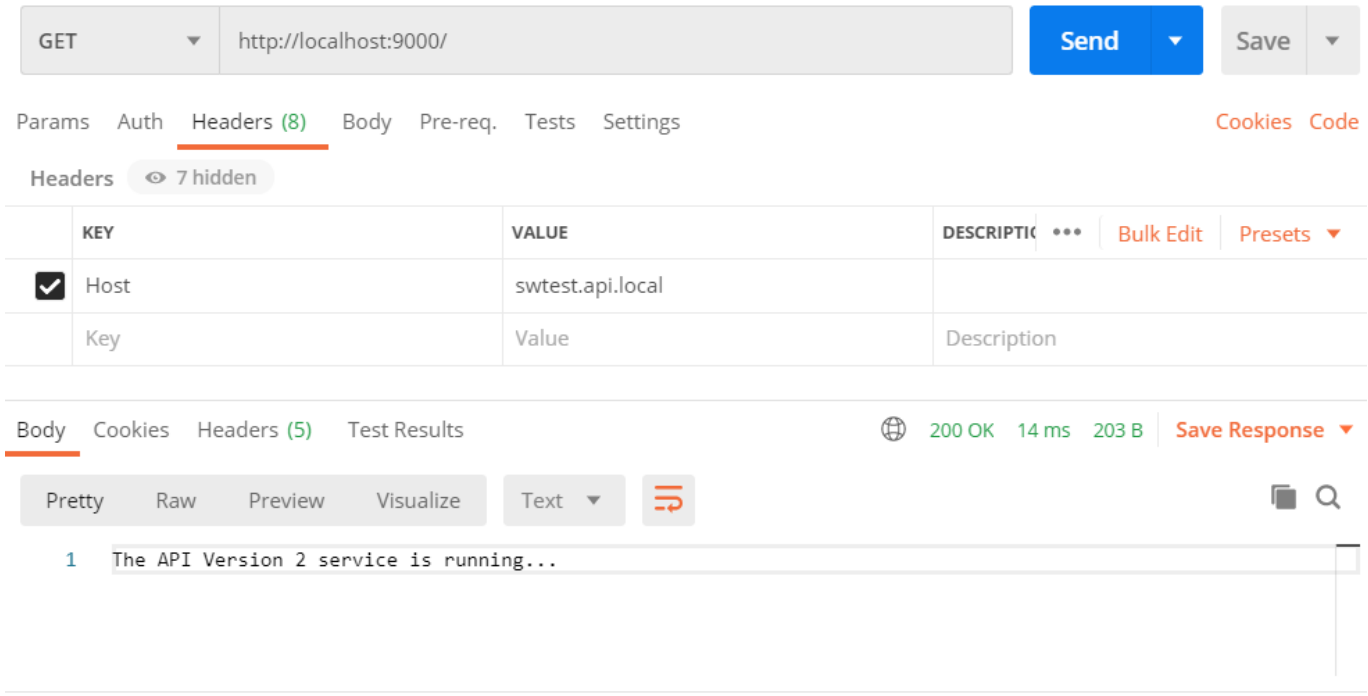
オンラインからpathを書き換えると即時にBlueからGreenにアプリケーションが切り替わる

新バージョンへのアクセス

Postmanで動作確認を行います

- URL: http://localhost:9000/
- Headers: Host -> swtest.api.local
- 「Send」ボタンクリック

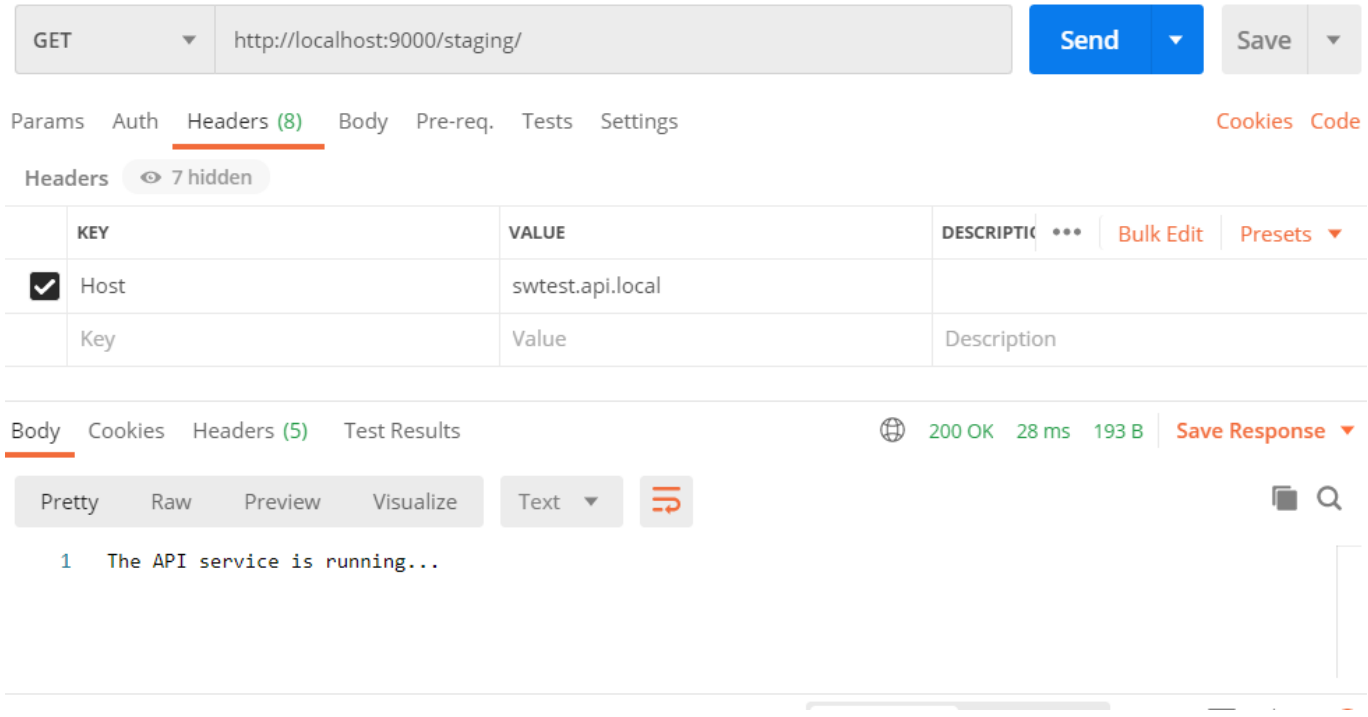
新バージョンの応答が返る



旧バージョンへのアクセス

- URL: http://localhost:9000/staging/
- Headers: Host -> swtest.api.local
- 「Send」ボタンクリック

旧バージョンの応答が返る



最後に実習環境の削除方法(できればKubernetes環境は破棄願います)

次回予定の、 `Apache Kafka` はDocker環境で実施します。Kubertenes環境が残っていると、動作が重くなります。
出来れば次回実習までにKubertenes環境を破棄願います。

PROXYのプロセスをCtrl+Cで終了させた後、構築の逆順でリソースを削除します。

Kubertenesリソース削除

1. Ingress: kubectl delete -f swtest-ingress.yaml
2. Service: kubectl delete -f swtest-service.yaml
3. Deployment: kubectl delete -f swtest-deployment-green.yaml
4. Deployment: kubectl delete -f swtest-deployment-blue.yaml

ingress-nginx, dashboard削除

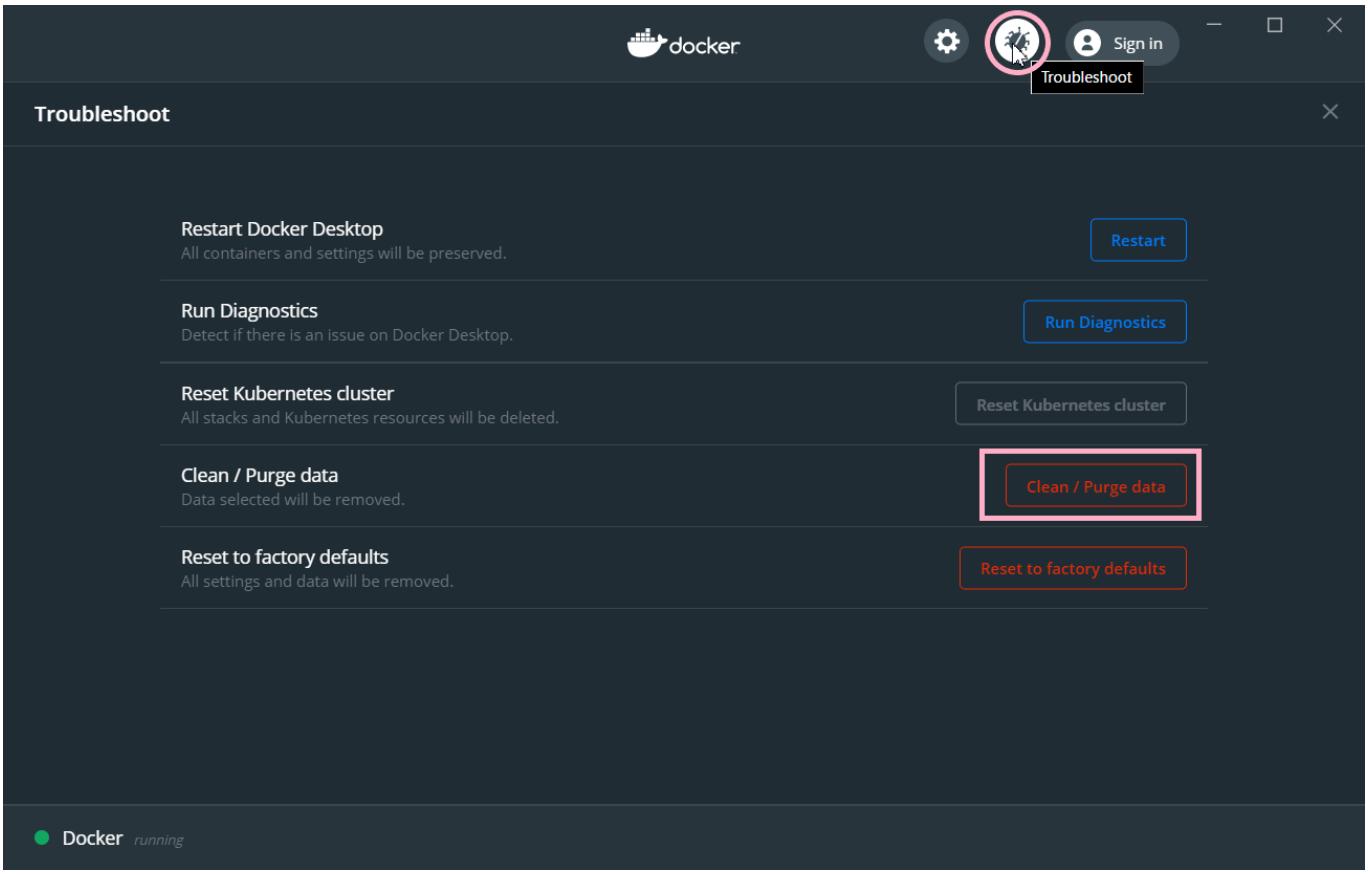
1. ingress-nginx: kubectl delete -f cloud-generic.yaml
2. ingress-nginx: kubectl delete -f mandatory.yaml
3. admin-user: kubectl delete -f role_binding.yaml
4. admin-user: kubectl delete -f create_account.yaml
5. dashboard: kubectl delete -f recommended.yaml

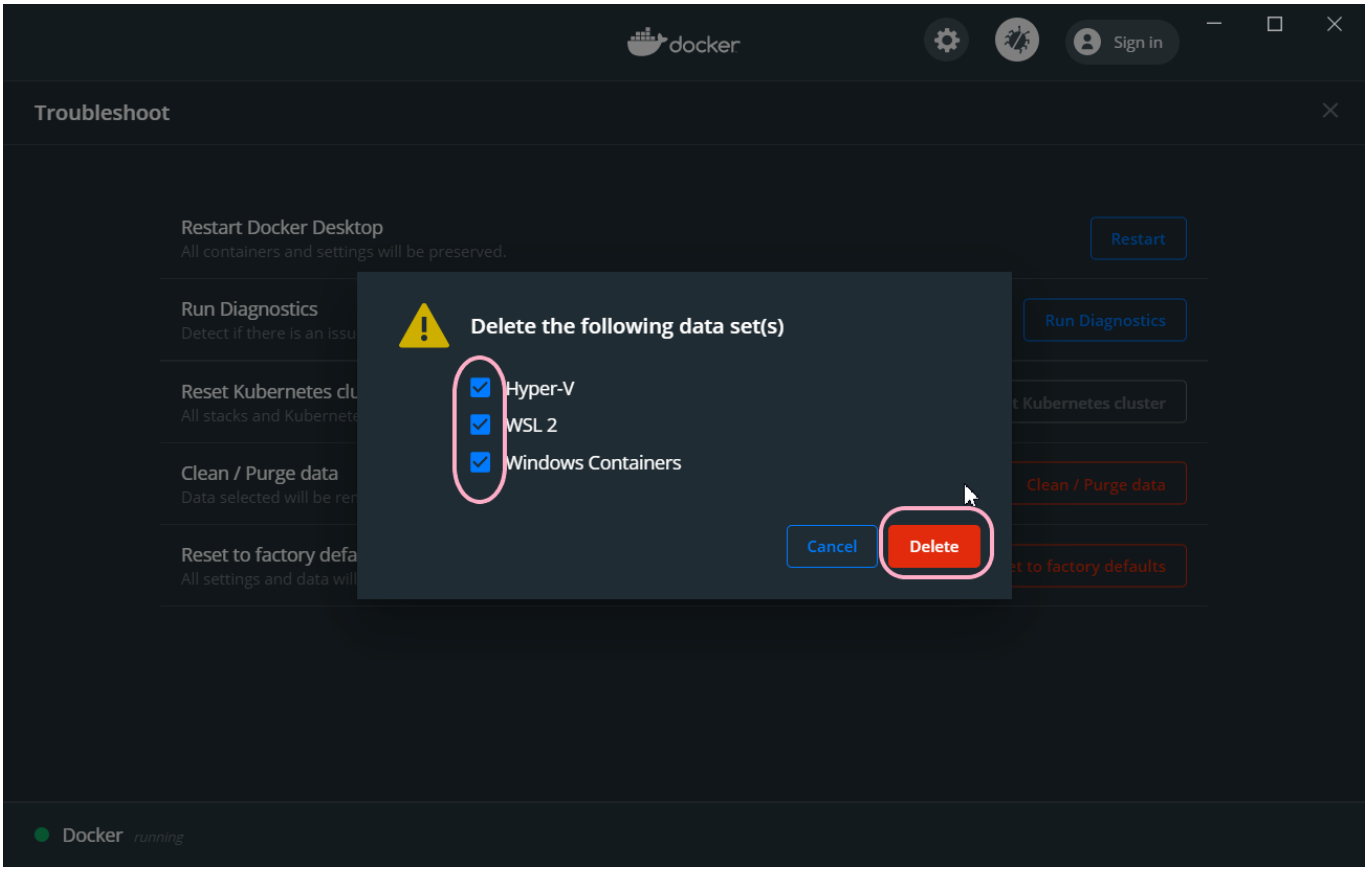
Docker Dashboardで無効化

1. Docker DashboardのSettingでKubertenesの有効化ボタンを元に戻す
2. 「Apply&Restart」ボタンを押す

不要データの削除

1. Docker DashboardのTroubleshootボタンを押す
2. 「Clean / Purge data」ボタンを押す
3. 削除対象のチェックを全て付ける
4. 「Delete」ボタンを押す


docker
Sign in
Troubleshoot
Troubleshoot
Restart Docker Desktop
All containers and settings will be preserved.
Restart
Run Diagnostics
Detect if there is an issue on Docker Desktop.
Run Diagnostics
Reset Kubernetes cluster
All stacks and Kubernetes resources will be deleted.
Reset Kubernetes cluster
Clean / Purge data
Data selected will be removed.
Clean / Purge data
Reset to factory defaults
All settings and data will be removed.
Reset to factory defaults
Docker running


docker
Sign in
Troubleshoot
Restart Docker Desktop
All containers and settings will be preserved.
Restart
Run Diagnostics
Run Diagnostics
Reset Kubernetes cluster
Clean / Purge data
Clean / Purge data
Reset to factory defaults
Delete the following data set(s)
Hyper-V
WSL 2
Windows Containers
Cancel
Delete
Docker running