Kubernetesを一言で言うと、自動デプロイ、スケーリング、アプリ・コンテナの運用自動化のために設計されたオープンソースのプラットフォームです。

Kubernetesによって、要求に迅速かつ効率良く対応ができます。

- アプリを迅速に予定通りにデプロイする (コンテナをサーバー群へ展開する)
- 稼働中にアプリをスケールする（稼働中にコンテナ数を変更する）
- 新機能をシームレスに提供開始する (稼働中にロールアウトする)
- ハードウェアの利用率を要求に制限する (コンテナで共存させて稼働率を高くする）

Kubernetesのゴールは、下記の様なアプリの運用負担を軽減するためのエコシステムのコンポーネントとツールを整備することです。

- 可搬性: パブリック・クラウド、プライベート・クラウド、ハイブリッド・クラウド、マルチ・クラウド
- 拡張可能: モジュール化、追加可能、接続可能、構成可能
- 自動修復: 自動配置、自動再起動、自動複製、自動スケーリング

2014年にプロジェクトが開始され、運用経験を基に、本番のワークロードを大規模に実行し、コミュニティのベストプラクティスのアイデアやプラクティスと組み合わせています。 Kubernetesの事例は https://kubernetes.io/case-studies/ にあります。

Kubernetesの発音は？

私の職場では、クバや、クーベルネティスと発音する人が多い感じがします。YouTubeで外人が発音しているのを聞くと、クーベネティスと言っている様に聞こえます。　発音について、Kubernetesの創設者の一人である@brendandburnsは、Twitterで次の様につぶやいています。

> @imdsm @gregde @francesc @jbeda @developerluke most people on the team say: koo-ber-net-ees, or just 'k8s' or k-eights
> — brendandburns (@brendandburns) 2015年4月7日

長いので、略してk8sという表記も多く使われています。

コンテナの利点を生かす

Kubernetesは、次の様なコンテナの利点を生かしたプラットフォームです。

- **アジャイルなアプリ開発とデプロイ：** 仮想マシンイメージの利用と比較して、コンテナ作成の簡便性や効率性が優れています。
- **継続的な開発、インテグレーション、デプロイ：** 信頼性の高く、頻繁なコンテナイメージのビルドとデプロイメントについて、迅速かつ簡単なロールバックを提供します。
- **DevとOpsの分離問題：** デプロイ時ではなくビルドとリリース時に、コンテナイメージを作ることで、アプリをインフラから分離できます。
- **開発、テスト、およびプロダクションの環境の一貫性：** ノートPCからクラウドでも同じように動作します。
- **クラウドとOSディストリビューションの可搬性：** Ubuntu, RHEL, CoreOS, オンプレ, Googleコンテナ・エンジン, そして、あらゆる場所で実行できます。
- **アプリ中心の管理：** アプリがOS上の論理的な資源(コンテナ)で動作するので、仮想マシン上のOSで動かすよりも抽象レベルが向上します。 抽象レベルが上がることで、単純化され、アプリに集中できます。
- **疎結合、分散、伸縮性、解放されたマイクロサービス：** アプリは、より小さい独立した部分に分割され、1つの大きな単一目的のマシン上で実行される大きなモノリシックな構造ではなく、動的に展開および管理できます。
- **資源の分離：** 予測可能なアプリのパフォーマンス
- **資源の稼働率：** 高い効率と密度

Kubernetesで何ができるか？

Kubernetesは、物理サーバーや仮想サーバーのクラスタ環境で、アプリのコンテナを実行します。そして、コンテナ中心の考え方によって、次の様なニーズに応えます。

- 共同化（１アプリ１コンテナを維持しながら、複合的なアプリを容易化) (自習ノート,IBM Cloud k8sの例)
- ストレージのマウント(IBM Cloud k8sの例)
- シークレットの分散 (パスワードなどを環境に保存します) (IBM Cloud k8sの例)
- アプリの健全性のチェック (コンテナのヘルスチェックを設定できます)
- アプリのインスタンスの複製 (自習ノート;デプロイメント・コントローラー)
- 水平の自動スケーリングの利用 (CPU稼働率の閾値でコンテナ数を増減します)
- サービス名と名前解決の管理 (サービスを登録すると、内部DNSで名前解決、環境変数を供給)
- ワークロードの分散 (自習ノート;サービス)
- ローリング・アップデート(自習ノート;ロールアウト)
- 資源の監視 (プロメテウス[Prometheus]とグラファナ[Grafana]が組み込まれています)
- ログの採取とアクセス (コンテナのSTDOUTを、Fluentd,ElasticSearch,Kibanaで集約して分析できます。ELK Stackは組み込まれています)
- アプリのデバッグ (デバッグをサポートする機能があります)
- 認証と認可の提供 (RBAC 役割ベースのアクセス制御とサービスアカウントでコントロールします)

ロールアウト（ローリング・アップデート）のデモ

Kubernetesの特徴的な利点を理解してもらえる様に、ロールアウトのムービーを作ってみました。

これは、アプリv1.0 の運用中に v1.1 へ ロールアウトする様子です。 画面は８つのブラウザをSeleniumから操作して、ログイン後に、ボタンをクリックする事でカウンターが更新する模擬アプリケーションが動作しています。 この状態からロールアウトを実行すると、セッションを維持しながら、次第にv1.1の画面へ変わっていきます。

Ver 1.0 運用中

ロールアウト中

Ver 1.1 ロールアウト完了

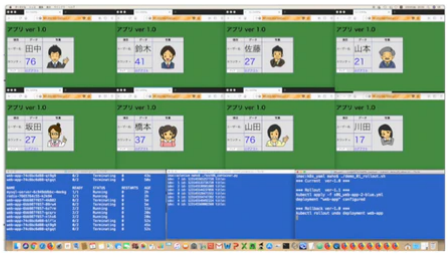
画面右下のターミナルがk8sの状態、画面真ん中下がselenium、画面左下がコマンド投入画面です。右下のターミナルを注目するとポッドが移行していく様子が確認できます。

つぎはは、ロールバックの進行中の様子です。画面右下のターミナルからロールバックのコマンドを投入後、次第に元のバージョンVer1.0へ戻っていく様子が伺えます。

Ver 1.1運用中

ロールバック中

ロールバック完了



こちらのデモは、https://youtu.be/ObA1OEVdrQY で参照できます。

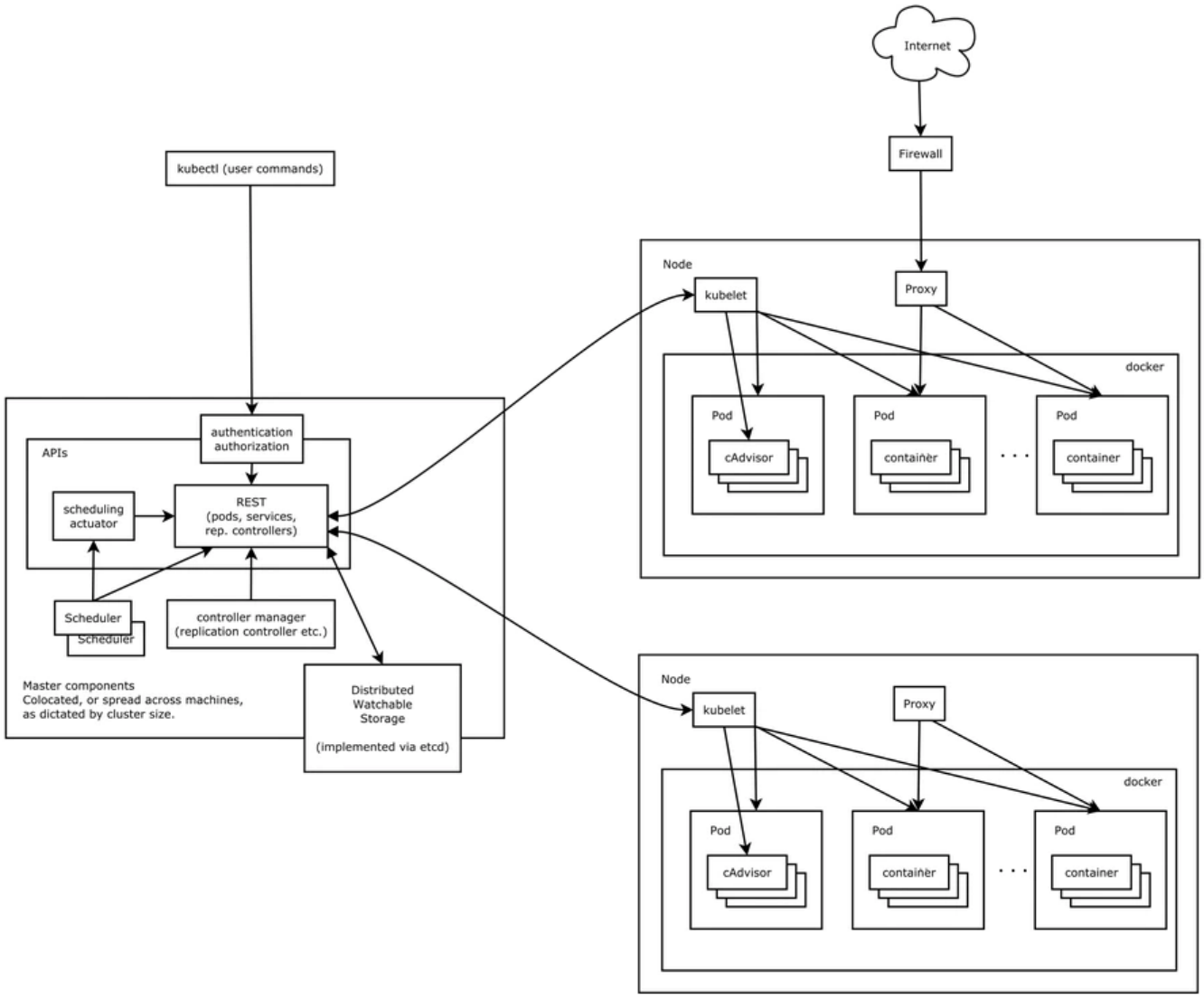
Kubernetes のアーキテクチャ

クーベネティスの構成要素すなわちアーキテクチャには、マスター・コンポーネントとノード・コンポーネントがあります。

- マスター・コンポーネントは、クラスタの制御を受け持ちます。 スケジューリングやイベントの検知や対応など、クラスタの全体に関わる決定を行います。　そして、クラスタ内のどのノードでも実行できます。

- ノード・コンポーネントは、すべてのノードで実行され、実行中のポッドを維持し、Kubernetesランタイム環境を提供します。

Kubernetesアーキテクチャ概要図 (https://github.com/kubernetes/kubernetes/blob/release-1.3/docs/design/architecture.md)



マスター・コンポーネント

- kube-apiserver: 制御部のフロントエンド
- etcd: 全てのクラスタデータ保存用のデータベース
- kube-controller-manager: ノード管理、レプリケーション管理、エンドポイント管理、サービスアカウントとトークン管理を実行

- cloud-controller-manager: 基盤となるクラウド・プロバイダーと連携、ノード、経路、サービス、ボリュームなどの作成・更新・削除を実施
- kube-scheduler: ポッドを監視して、ノードを実行する場所を選択する
- addons: アドオンは、クラスタ機能を実装するポッドとサービスです。
  - DNS: クーベネティスのサービスを探すために必須のDNSサーバー
  - User interface: kube-uiは、表示だけのUIです。
  - Container Resource Monitoring: 時系列のデータを記録して、参照のUIを提供
  - Cluster-level Logging: コンテナのログの集中保管、検索、表示を提供

ノード・コンポーネント

- kubelet: プライマリのエージェントで、ノードに割り当てられたポッドを監視
- kube-proxy: ネーミングルールに基づき接続や転送を実施
- docker: コンテナの実行に利用
- rkt: 実験的なDockerの代替
- supervisord: kubelet と dockerのプロセス監視と制御
- fluentd: ログ取得のためのプロセス

マスター・ノードとワーカー・ノード

マスター・コンポーネントを実行するサーバーは、マスター・ノードと呼ばれます。または、単にマスターとも呼ばれます。また、ノード・コンポーネントだけを実行するサーバーをワーカーノードと呼びます。こちらは、単にノードと呼ばれることもあります。

学習用環境 Minikube では、マスター・コンポーネントとノード・コンポーネントを、一つのサーバーの中で実行してコンパクトな環境を作ります。

マスター・コンポーネントを実行するサーバーに、ノード・コンポーネントの実行は必須ではありません。Kubernetes the hard way では、マスター・コンポーネントは、３つの仮想サーバー上にクラスタ化して配置する方法が提示されています。この構成では、マスター・ノードは、ノードの一つとして構成されないため、kubectl get node で管理することができませんが、マスターノードをマネージドサービスとして提供するパブリック・クラウドでは好まれる構成です。

一方、kubeadmコマンドを利用してクラスタを構成した場合、ノードの一つにマスター・コンポーネントを配置しますから、ノードの一つとしてマスター・ノードを管理することができます。

各社クラウド・プロバイダー間の互換性

Kubernetesの適合性認定を各社取得していると思うが、どれくらい互換性があるのか、疑問だったので調べて見ました。

確認対象は、マルチゾーンが設定できる グーグル GKE, アマゾン EKS, アイ・ビー・エム IKS について調べました。簡単に結果を言うと、同じマニフェスト(YAML)で 3 つのクラウドで動作しました。しかも、修正無しでした。 ただし、クラウドのIaaS部分は、gcloudコマンド、awsコマンド、ibmcloudコマンドなど、各社各様だったので、その部分に苦労しましたが、解ってしまえば簡単です。

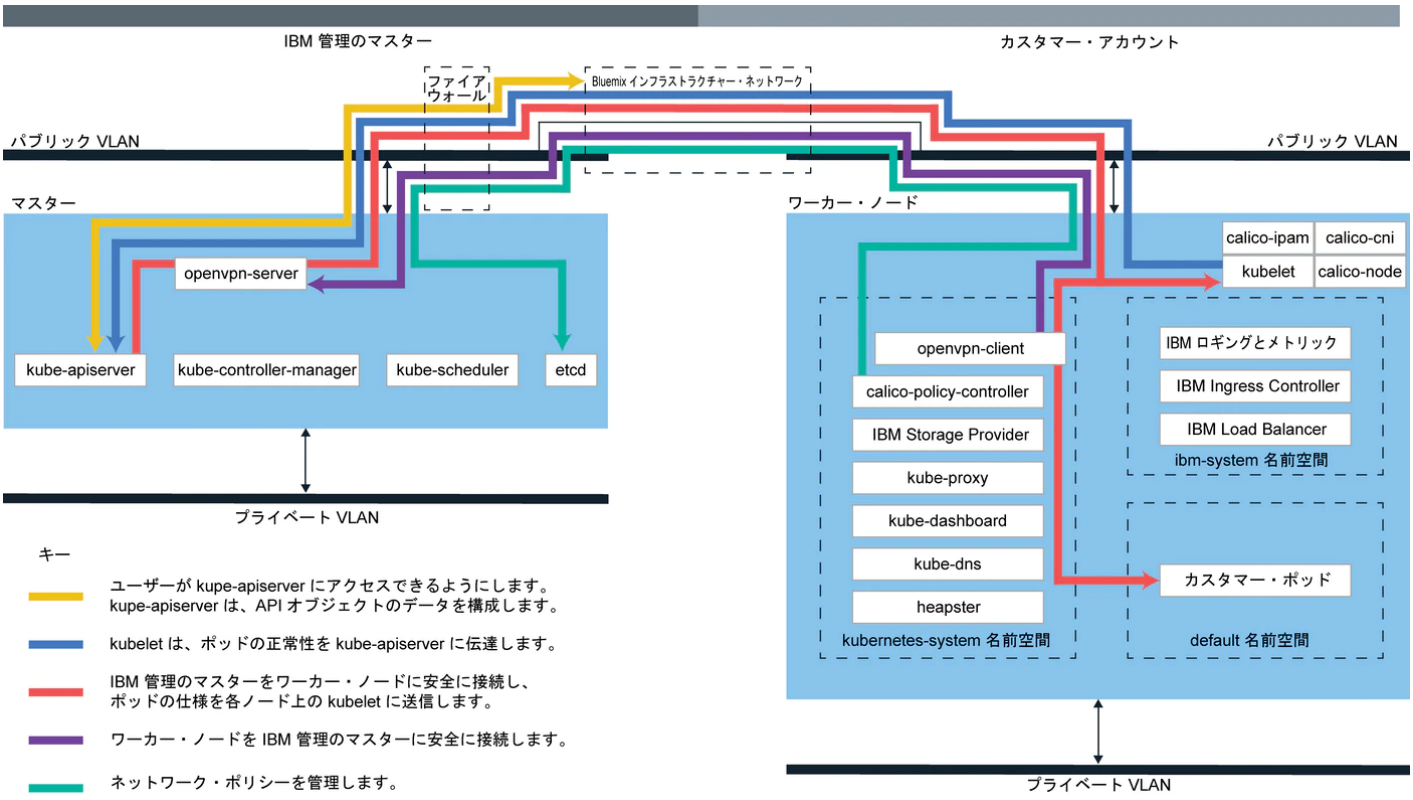
- AWS初心者だけど、EKSのクラスタを作って、NodePort と ELBアクセス、EBSのPVを確認したよ
- GCP初心者が、GKEクラスタを作って、NodePort、ロードバランサー、永続ボリュームを使ってみたよ。
- EKSやGKEで動作確認したマニフェストをIKSで動かした結果

個人的な感想としては、以下のような感じです。

- 簡単に利用できた順　1. IKS, 2. GKE, 3. EKS
- 機能が豊富な順　1. EKS, 2. GKE, 3. IKS
- 簡単さと高機能を兼ね備えているのは GKE、さすがグーグルという感じでした。
- 各社ともメトリックスとログ監視が、クラウドに依存した方法なので、マルチクラウドを目指すのであれば、検討が必要と思います。

IBM Cloud での実装

前述のマスタ・コンポーネントは、マスターとワーカー・ノードに分散してデプロイされます。そして、ノード・コンポーネントは、ワーカー・ノードにデプロイされます。 マスターは、IBM 管理として提供され、ユーザーが直接ログインすることはできません。 一方、ワーカー・ノードは、IBM Cloud Infrastructure の仮想サーバーとしてプロビジョニングされます。



IBM Cloud コンソール画面のワーカー・ノードのリスト

専用のワーカー・ノード数を 3 個として、Kubernetes クラスタを起動した時の画面です。

このワーカー・ノードは、Bluemix Infrasttucre の仮想サーバーとして起動されているので、IBM Cloud Infrastructure 側からの画面で、３台の仮想サーバーとして確認することができます。

まとめ

Kubernetes のコマンドを実行したり設定する記事は、たくさんあるのですが、なんのため？、何ができるの？、利点は？　といった言葉の説明が無いので、参考資料を読みながら、まとめてみました。

参考資料

(1) What is Kubernetes https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
(2) What is Kubernetes https://www.youtube.com/watch?v=R-3dfURb2hA
(3) Kubernetes Components https://kubernetes.io/docs/concepts/overview/components/
(4) Kubernetes クラスターと IBM Bluemix Container Service について https://console.bluemix.net/docs/containers/cs_ov.html#cs_ov