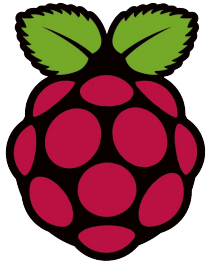
kubeadmよりも遠い場所

~手動でガンバるKubernetes on Raspberry Pi~



自己紹介 1人目

- 所属
  - 公立はこだて未来大学
  - システムソフトウェア研究室
- 趣味
  - ゲーム(Fire Emblem)
  - マンガ(Girls Love)
- 使ってる言語
  - Go言語

永井陽太

自己紹介 2人目

高橋 大輔

公立はこだて未来大学

- システムソフトウェア研究室
  - 高度ICTコース 4年
- 学内クラウドチーム

趣味

- 読書 &（時々ノープランで旅行）

もくじ

- Kubernetesってなぁに？
- Kubernetesを利用するために
- Kubernetesのコンポーネントと役割
- Let’s 構築！！

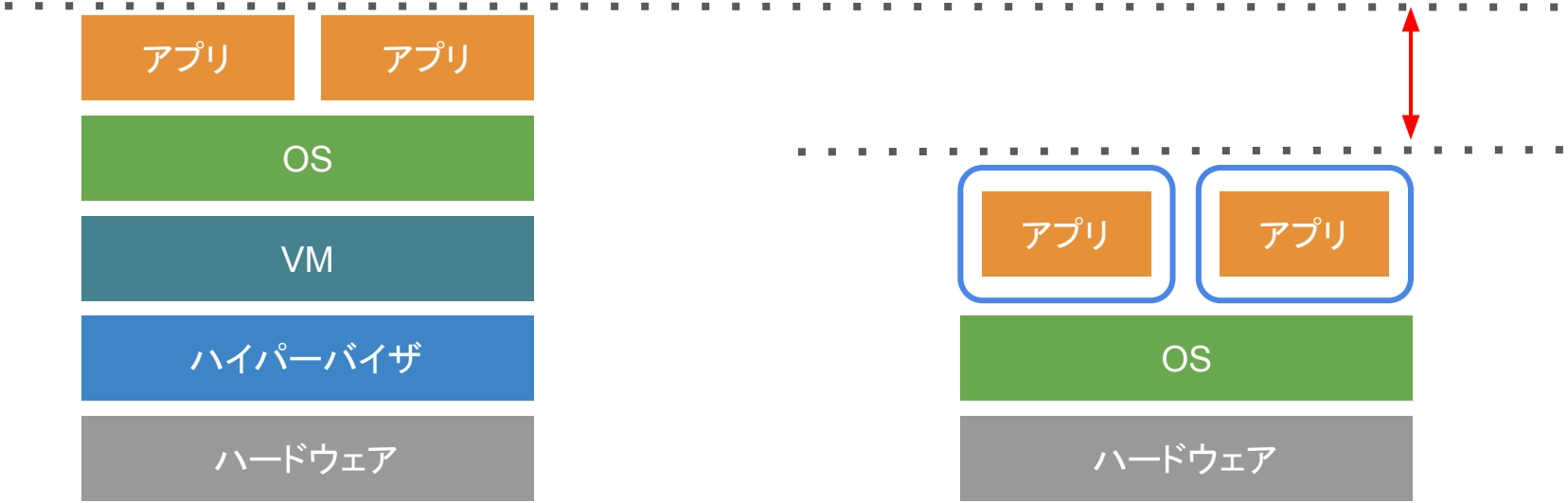
Kubernetesってなぁに？

Kubernetesとは

- 公式サイトによると
  - コンテナ化されたサービスを管理するための移植性のある拡張可能なオープンソースプラットフォーム
- 要するに
  - 便利なコンテナ管理のプラットフォーム！！

ちなみに

- コンテナとは
  - 従来の仮想化よりも軽量な仮想化

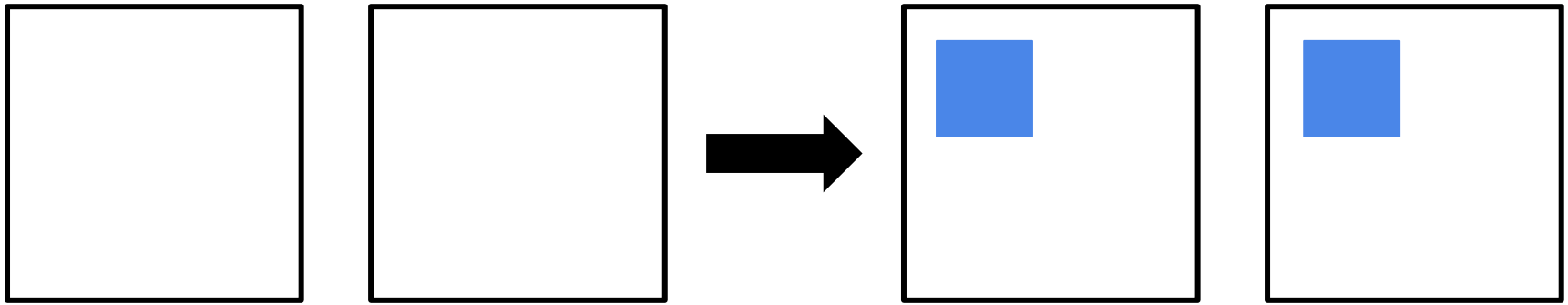


Kubernetesがやってくれること

- 複数ホスト上にコンテナを展開
- コンテナの死活監視とセルフヒーリング
- コンテナの負荷分散

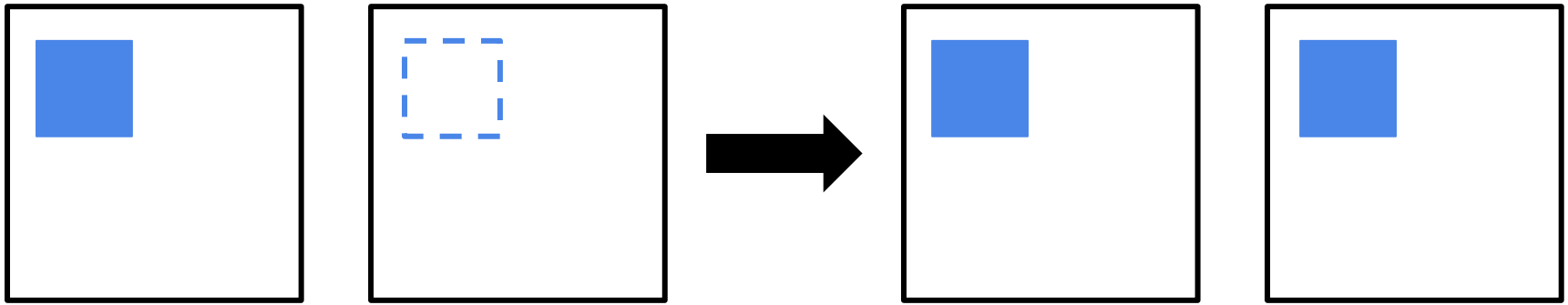
Kubernetesがやってくれること

- **複数ホスト上にコンテナを展開**
- コンテナの死活監視とセルフヒーリング
- コンテナの負荷分散



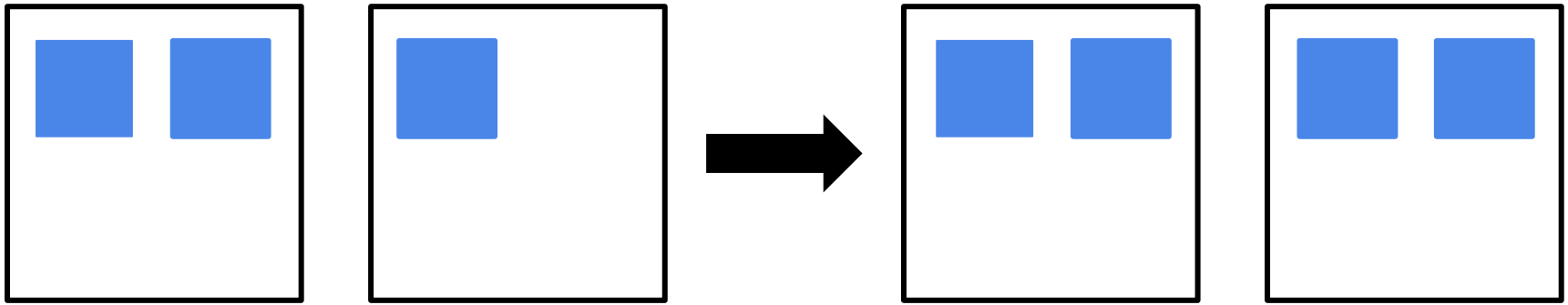
Kubernetesがやってくれること

- 複数ホスト上にコンテナを展開
- **コンテナの死活監視とセルフヒーリング**
- コンテナの負荷分散



Kubernetesがやってくれること

- 複数ホスト上にコンテナを展開
- コンテナの死活監視とセルフヒーリング
- **コンテナの負荷分散**



利用が広がるKubernetes

- 主要パブリッククラウドにてKubernetesを利用したサービスが展開されている
  - GCP：Google Kubernetes Engine(GKE)
  - AWS：Amazon Elastic Container Service for Kubernetes(EKS)
  - Azure：Azure Kubernetes Service(AKS)
- Kubernetesの利用はこれからもどんどん広がっていく

Kubernetesを利用するために

さまざまな利用方法

- クラウドサービス
  - GCP
  - AWS
  - Azure
- オンプレミス
  - クラスタ構築支援ツール
  - 手動

さまざまな利用方法

- クラウドサービス
  - GCP
  - AWS
  - Azure
- オンプレミス ←今日話すこと
  - クラスタ構築支援ツール
  - 手動

構築支援ツール その1

- kubeadm
  - Kubernetesのクラスタ構築に必要なコンポーネントの起動をすべて自動で行う
  - 基本的に以下のコマンドで終わり
    - kubeadm init
    - kubeadm join
  - とっても簡単!!
  - でも，うまくいかないことが多い(^_^;)

構築支援ツール その2

- hyperkube
  - 必要なコンポーネントすべてを含んだバイナリ
  - コマンドライン引数にコンポーネント名を指定すると
    そのコンポーネントを起動する
  - 全部入っているため便利で楽ちん!!

でも，簡単に楽に終わったらつまらない

手動構築という選択

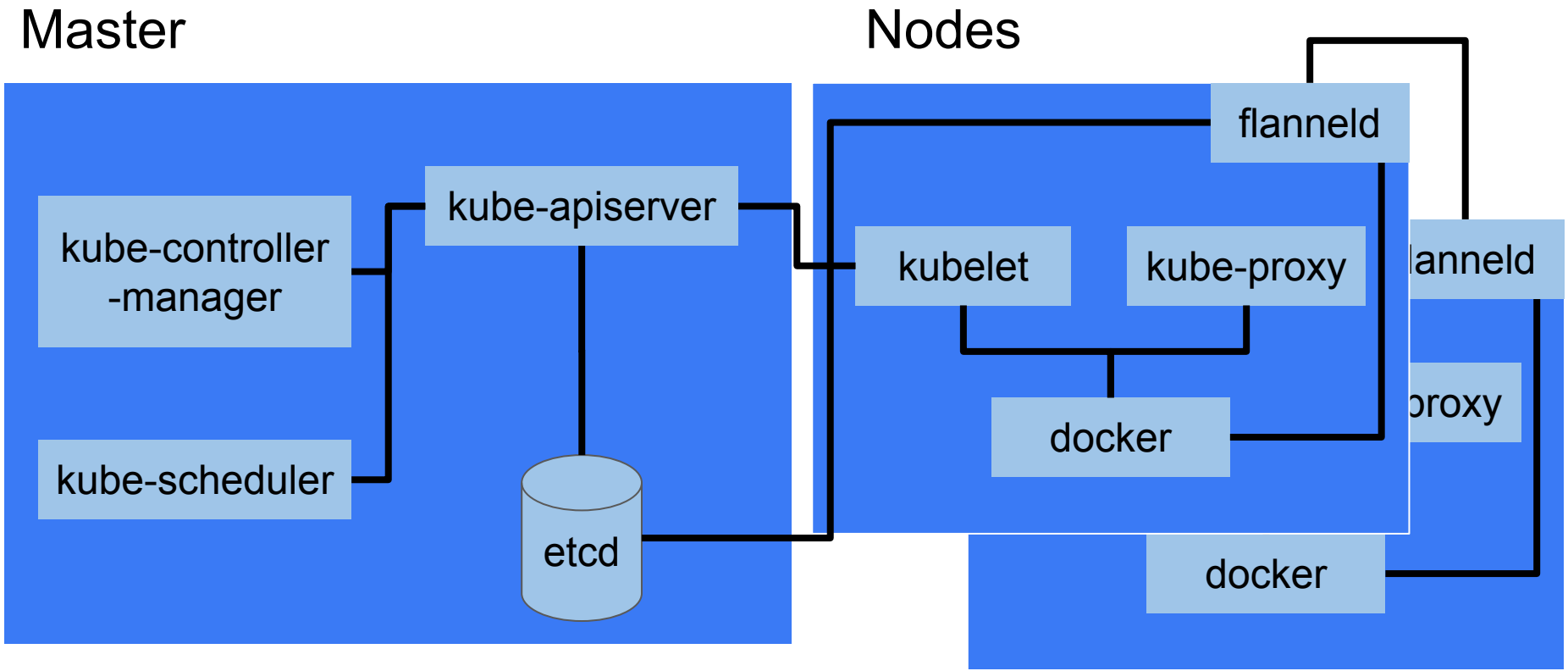
- Kubernetesのコンポーネントをすべて自分でビルド！
  - CPUのアーキテクチャに合わせたKubernetesクラスタの構築が可能に
  - 今回はARM
- Kubernetesのコンポーネントを自分で起動！
  - 設定ファイルからUnitファイルまで
    すべて自分で記述することで
    Kubernetesの仕組みがよく分かる

Kubernetesのコンポーネントと役割

Kubernetesのコンポーネント

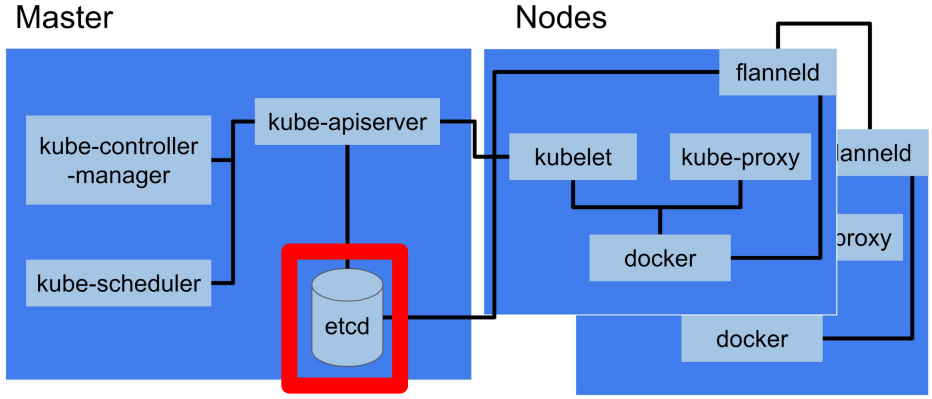
- Master
  - etcd
  - kube-apiserver
  - kube-scheduler
  - kube-controller-manager
  - flanneld
- Node
  - kubelet
  - kube-proxy
  - flanneld

Kubernetesクラスターの構成



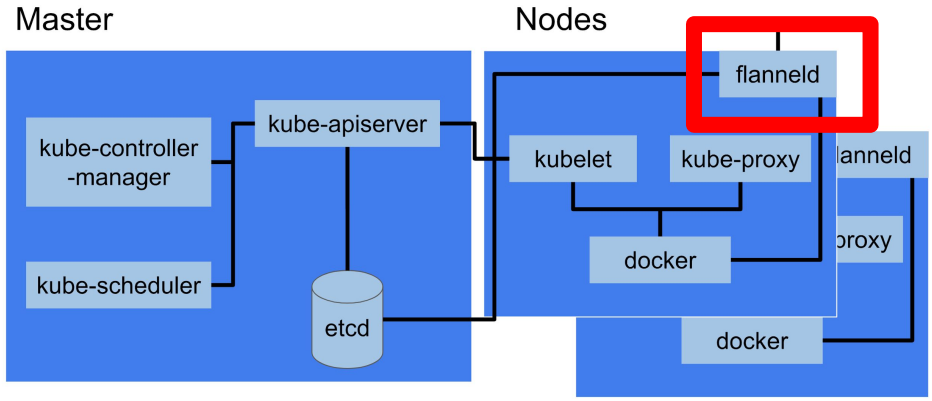
etcd

- etcdとは
  - KVS(Key Value Store)
  - 一意なKeyと対応させてValueを保存
- Kubernetesにおける役割
  - Kubernetesクラスターの情報を保存する



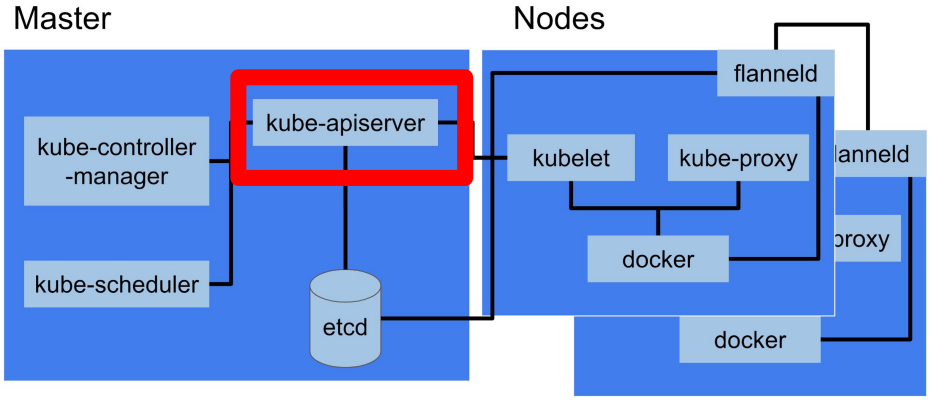
flanneld

- flanneldとは
  - ホストを超えてコンテナ同士が通信するための内部ネットワークを提供するFlannelのデーモン



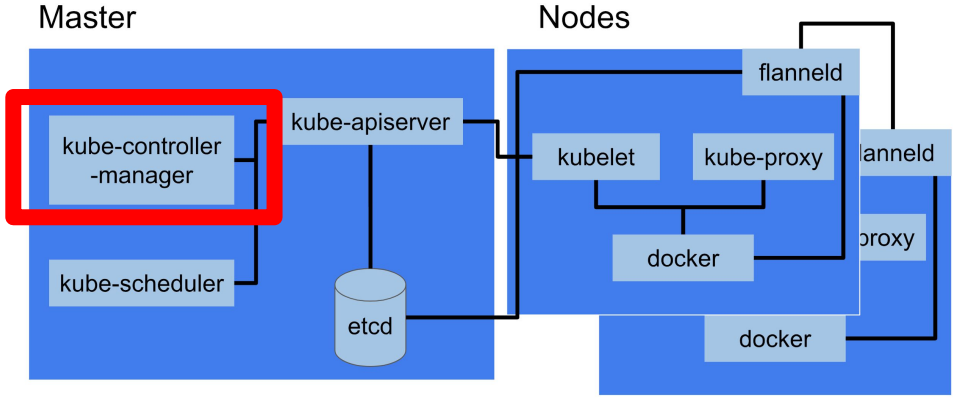
kube-apiserver

- Kubernetes内のリソースを操作するためのAPIサーバ
- すべての操作はこのAPIサーバを通じて行う



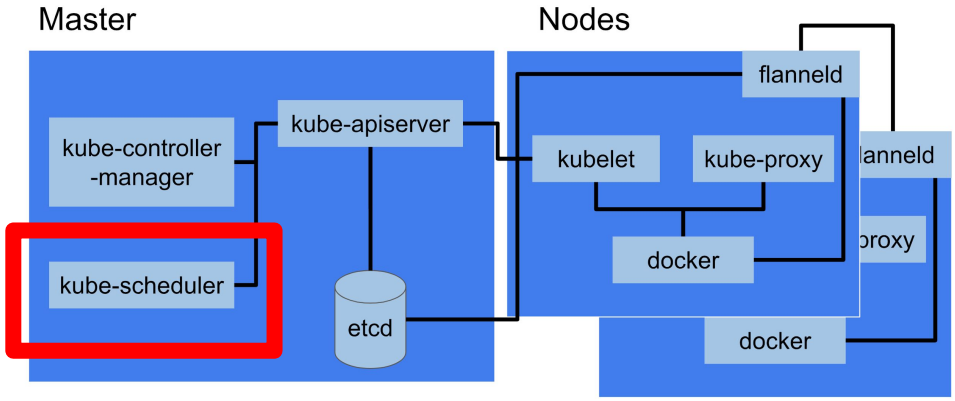
kube-controller-manager

- Kubernetes上の各種リソースを管理するコントローラを起動するデーモン
  - replication controller
  - replicaset controller
  - deployment controller
  - etc...



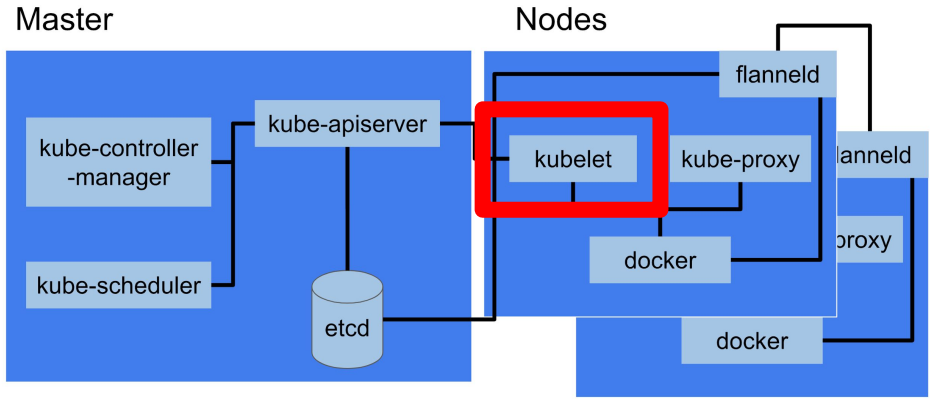
kube-scheduler

- 新しく作成されたコンテナをどのNodeに割り当てるのか決定するデーモン



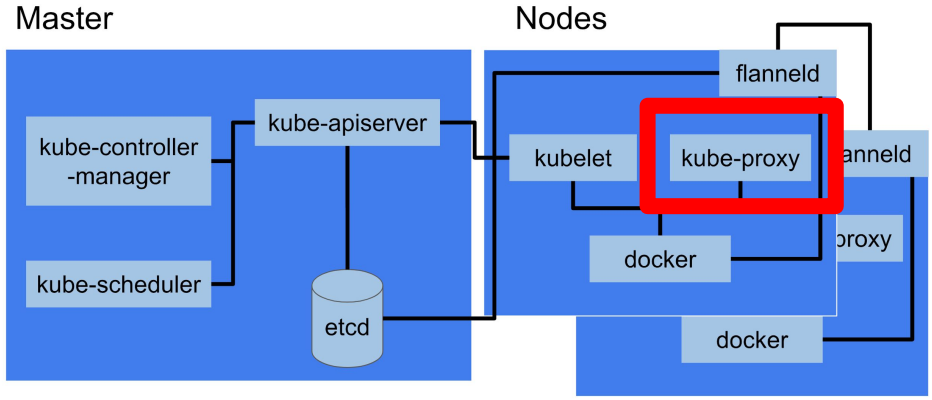
kubelet

- kube-apiserverの指示をうけて，Node上にコンテナを作成したり，削除したりするデーモン



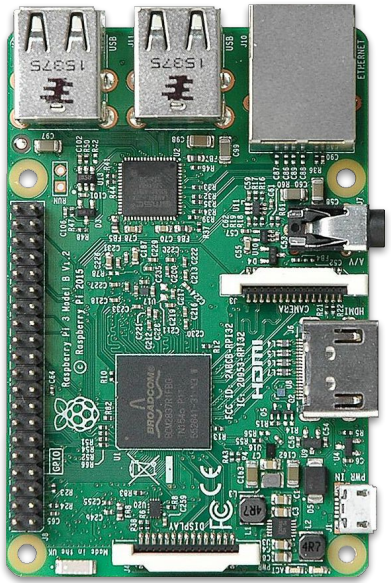
kube-proxy

- iptablesやIPVSと連携し，Node内の
  コンテナに対してパケットの転送を行うデーモン



今回の構成 (1/2)

- 機材
  - Raspberry Pi 3 Model B x3
- ソフトウェア
  - Raspbian: Stretch Lite
  - Kubernetes: ~~1.11.0~~
    - 32bitでの動作にバグあり
    - release-1.11ブランチの最新版をビルド
  - etcd: 3.3.8
  - Flannel: 0.10.0
  - CNIプラグイン: 0.7.1



今回の構成 (2/2)

- 1台目のみ (master)
  - etcd, kube-apiserver, kube-scheduler, kube-controller-manager
- 3台に共通 (node)
  - Docker, kubelet, kube-proxy, Flannel

手順

0. 準備
1. ビルド
2. バイナリの配置
3. masterのセットアップ
4. nodeのセットアップ
5. (最後に、何かをデプロイしてみる)

手順

0. 準備
1. ビルド
2. バイナリの配置
3. masterのセットアップ
4. nodeのセットアップ
5. (最後に、何かをデプロイしてみる)

準備

- swapの無効化
  - $ sudo systemctl disable dphys-swapfile.service
- Cgroupの設定
  - /boot/cmdline.txtに”cgroup_enable=memory cgroup_memory=1”を追記

以下はお好みで。

- Kernelを64bitでビルドし直す
- オーバークロック (1.2GHz → 1.3GHz)
  - ヒートシンクをお忘れなく

手順

0. 準備
1. ビルド
2. バイナリの配置
3. masterのセットアップ
4. nodeのセットアップ
5. (最後に、何かをデプロイしてみる)

ビルド

ビルド環境

- Google Cloud PlatformのCompute Engineを利用
- マシンタイプ: n1-standard-96
  - CPU: 96コア
  - メモリ: 360GB
  - **Preemptible VMでお安く!** (1/4くらい)
- OS: Ubuntu 18.04 64bit
  - ディスク: SSD PD 100GB

⇒ $0.983/hour × 約2時間 ≒ 220円

ビルド - Kubernetes

```
$ sudo apt update
$ sudo apt install -y docker.io make
$ sudo usermod -a -G docker ${USER}
$ git clone https://github.com/kubernetes/kubernetes.git
$ cd kubernetes
$ git checkout release-1.11
$ make release # _output/release-tars/kubernetes-server-linux-arm.tar.gzに吐き出されている
```

(所要時間=約45分, ディスク消費=約70GB)

ビルド - Kubernetes (未検証版)

```
$ sudo apt update
$ sudo snap install --classic go
$ go get -u github.com/jteeuwen/go-bindata/...
$ sudo apt install gcc-arm-linux-gnueabihf
$ git clone https://github.com/kubernetes/kubernetes.git
$ cd kubernetes
$ git checkout release-1.11
$ make all WHAT=cmd/kubectl KUBE_VERBOSE=5 KUBE_BUILD_PLATFORMS=linux/arm
$ make all WHAT=cmd/kube-apiserver KUBE_VERBOSE=5 KUBE_BUILD_PLATFORMS=linux/arm
$ make all WHAT=cmd/kube-scheduler KUBE_VERBOSE=5 KUBE_BUILD_PLATFORMS=linux/arm
$ make all WHAT=cmd/kube-controller-manager KUBE_VERBOSE=5 KUBE_BUILD_PLATFORMS=linux/arm
$ make all WHAT=cmd/kube-proxy KUBE_VERBOSE=5 KUBE_BUILD_PLATFORMS=linux/arm
$ make all WHAT=cmd/kubelet KUBE_VERBOSE=5 KUBE_BUILD_PLATFORMS=linux/arm
```

ビルド - etcd

```
$ sudo snap install --classic go
$ git clone https://github.com/coreos/etcd.git
$ cd etcd
$ git checkout v3.3.8
$ GOOS=linux GOARCH=arm ./build # bin/{etcd,etcdctl}に吐き出されている
```

参考: [Building etcd]

ビルド - flannel

```
$ sudo snap install --classic go
$ sudo apt install qemu-arm-static
$ git clone https://github.com/coreos/flannel.git
$ cd flannel
$ git checkout v0.10.0
$ make qemu-arm-static # クロスビルド時のみ
$ make dist/flanneld-arm # dist/{flanneld-arm}に吐き出されている
```

参考: Building flannel

ビルド - CNIプラグイン

```
$ sudo snap install --classic go
$ git clone https://github.com/containernetworking/plugins.git
$ cd plugins
$ git checkout v0.7.1
$ GOOS=linux GOARCH=arm ./build.sh # bin以下に吐き出されている
```

バイナリの配置

0. 各々をRaspberry Piに転送
1. kubernetes-server-linux-arm.tar.gzを展開
2. /usr/bin/以下にKubernetes, etcd, flannelのバイナリを配置
3. (systemdのUnitファイル・設定ファイルを配置)
    - 参考: 学内向けに書いたQiita, Kubernetes , etcd, flannel (簡略化のため、一部を編集)
4. /opt/cni/bin/以下にCNIプラグインのバイナリを配置

手順

0. 準備
1. ビルド
2. バイナリの配置
3. masterのセットアップ
4. nodeのセットアップ
5. (最後に、何かをデプロイしてみる)

セットアップ - master

master: 証明書関連

公式ドキュメントを参考に、easyrsaで自己署名の証明書を発行

- MasterのCluster IPは推測するしかない (きっと10.0.0.1)
- 生成できた証明書は /etc/kubernetes/tls/に移動

確認: {ca,server}.{crt,key}が生成されていること

master: etcd

設定を追加 (/etc/etcd/etcd.conf)

```
ETCD_UNSUPPORTED_ARCH=arm
ETCD_LISTEN_CLIENT_URLS="http://0.0.0.0:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://0.0.0.0:2379"
```

確認: etcdctl cluster-healthでエラーがないこと

master: kube-apiserver

設定を追加・変更 (/etc/kubernetes/apiserver)

- --insecure-bind-address=0.0.0.0
- --service-cluster-ip-range=10.0.0.0/16
- --client-ca-file=/etc/kubernetes/tls/ca.crt
- --tls-cert-file=/etc/kubernetes/tls/server.crt
- --tls-private-key-file=/etc/kubernetes/tls/server.key

設定を変更 (/etc/kubernetes/config)

- --master={masterのIP}

確認: kubectl get nodesでエラーがないこと (結果は0件)

※kubectl config {set-cluster, set-context, use-context}が必要

master: kube-apiserver

```
# KUBELET_PORT="--kubelet-port=10250"

# Comma separated list of nodes in the etcd cluster
KUBE_ETCD_SERVERS="--etcd-servers=http://127.0.0.1:2379,http://127.0.0.1:4001"

# Address range to use for services
KUBE_SERVICE_ADDRESSES="--service-cluster-ip-range=10.0.0.0/16"

# default admission control policies
KUBE_ADMISSION_CONTROL="--admission-control=NamespaceLifecycle,LimitRanger,SecurityContextDeny,ServiceAccount,ResourceQu
ota"

# Add your own!
KUBE_API_ARGS="--client-ca-file=/etc/kubernetes/tls/ca.crt --tls-cert-file=/etc/kubernetes/tls/server.crt --tls-private-
key-file=/etc/kubernetes/tls/server.key"
```

master: kube-apiserver

```
###
# kubernetes system config
#
# The following values are used to configure various aspects of all
# kubernetes services, including
#
#   kube-apiserver.service
#   kube-controller-manager.service
#   kube-scheduler.service
#   kubelet.service
#   kube-proxy.service
# logging to stderr means we get it in the systemd journal
KUBE_LOGTOSTDERR="--logtostderr=true"

# journal message level, 0 is debug
KUBE_LOG_LEVEL="--v=0"

# Should this cluster be allowed to run privileged docker containers
KUBE_ALLOW_PRIV="--allow-privileged=false"

# How the controller-manager, scheduler, and proxy find the apiserver
KUBE_MASTER="--master=http://203.104.213.137:8080"
```

master: kube-apiserver

```
root@k8s-test:~# kubectl config set-cluster default --server=http://203.104.213.137:8080
Cluster "default" set.
root@k8s-test:~# kubectl config set-context default --cluster=default
Context "default" modified.
root@k8s-test:~# kubectl config use-context default
Switched to context "default".
root@k8s-test:~# kubectl get nodes
root@k8s-test:~#
```

master: kube-scheduler

確認: systemctl status kube-scheduler (起動後しばらくは要チェック)

master: kube-controller-manager

設定を追加 (/etc/kubernetes/controller-manager)

- --service-account-private-key-file=/etc/kubernetes/tls/server.key

確認: systemctl status kube-controller-manager (起動後しばらくは要チェック)

```
###
# The following values are used to configure the kubernetes controller-manager

# defaults from config and apiserver should be adequate

# Add your own!
KUBE_CONTROLLER_MANAGER_ARGS="--service-account-private-key-file=/etc/kubernetes/tls/server.key"
```

手順

0. 準備
1. ビルド
2. バイナリの配置
3. masterのセットアップ
4. nodeのセットアップ
5. (最後に、何かをデプロイしてみる)

セットアップ: node

セットアップ: node

今回は、まず1台目をセットアップ

その後、2, 3台目も同様に行います

node: Docker

curl -sSL https://get.docker.com | sudo shでインストール

設定を追加 (/lib/systemd/system/docker.service)

- --exec-opt native.cgroupdriver=systemd

確認: docker infoでCgroup Driverがsystemdになっていること

node: Docker

```
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target docker.socket firewalld.service
Wants=network-online.target
Requires=docker.socket

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --exec-opt native.cgroupdriver=systemd
ExecReload=/bin/kill -s HUP $MAINPID
LimitNOFILE=1048576
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNPROC=infinity
LimitCORE=infinity
# Uncomment TasksMax if your systemd version supports it.
# Only systemd 226 and above support this version.
TasksMax=infinity
TimeoutStartSec=0
# set delegate yes so that systemd does not reset the cgroups of docker containers
Delegate=yes
# kill only the docker process, not all processes in the cgroup
KillMode=process
# restart the docker process if it exits prematurely
Restart=on-failure
StartLimitBurst=3
StartLimitInterval=60s

[Install]
WantedBy=multi-user.target
```

node: flannel

設定を追加 (/etc/flanneld/flanneld)

- --etcd-endpoints=http://{master-ip}:2379
- --etcd-prefix=/kubernetes/network

コマンドを実行

```
$ etcdctl set /kubernetes/network/config
‘{ "Network": "10.0.0.0/16", "Backend": {
"Type": "vxlan" } }’
```

/etc/cni/net.d/10-flannel.confを作成

```
{
  "name": "podnet",
  "type": "flannel",
  "delegate": { "isDefaultGateway": true }
}
```

確認: /var/run/flannel/subnet.envが正しく作成されていること

node: flannel

```
FLANNEL_OPTS="--etcd-endpoints=http://203.104.213.137:2379 --etcd-prefix=/kubernetes/network"
```

```
root@k8s-test:~# etcdctl set /kubernetes/network/config '{ "Network": "10.0.0.0/16", "Backend": { "Type": "vxlan" } }'
{ "Network": "10.0.0.0/16", "Backend": { "Type": "vxlan" } }
root@k8s-test:~#
```

```
{
    "name": "podnet",
    "type": "flannel",
    "delegate": {
        "isDefaultGateway": true
    }
}
```

```
FLANNEL_NETWORK=10.0.0.0/16
FLANNEL_SUBNET=10.0.88.1/24
FLANNEL_MTU=1404
FLANNEL_IPMASQ=false
```

node: kube-proxy

確認: systemctl status kube-proxy (起動後しばらくは要チェック)

node: kubelet

kubectlから設定をエクスポート

```
$ kubectl config view > /etc/kubernetes/kubelet.kubeconfig
```

設定を追加 (/etc/kubernetes/kubelet)

- --address={0.0.0.0 or 使用するインターフェイスの IP}
- --network-plugin=cni
- --cni-conf-dir=/etc/cni/net.d
- --cni-bin-dir=/opt/cni/bin
- --cluster-dns 1.1.1.1, 8.8.8.8 # 適当なDNSサーバを指定

確認: kubectl get nodesの結果にmaster1台のみが表示されている

node: kubelet

```
###
# kubernetes kubelet (minion) config

# The address for the info server to serve on (set to 0.0.0.0 or "" for all interfaces)
KUBELET_ADDRESS="--address=0.0.0.0"

# The port for the info server to serve on
# KUBELET_PORT="--port=10250"

# You may leave this blank to use the actual hostname
KUBELET_HOSTNAME=""

# Edit the kubelet.kubeconfig to have correct cluster server address
KUBELET_KUBECONFIG="--kubeconfig=/etc/kubernetes/kubelet.kubeconfig"

# Add your own!
KUBELET_ARGS="--cgroup-driver=systemd --fail-swap-on=false --network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-
dir=/opt/cni/bin --cluster-dns 1.1.1.1,8.8.8.8"
```

```
root@k8s-test:~# kubectl get nodes
NAME       STATUS    ROLES     AGE       VERSION
k8s-test   Ready     <none>    6d        v1.11.0
root@k8s-test:~#
```

手順

0. 準備
1. ビルド
2. バイナリの配置
3. masterのセットアップ
4. nodeのセットアップ
5. (最後に、何かをデプロイしてみる)

デプロイ

Nginxのデプロイ

kubectlからNginxをデプロイ

```
$ kubectl apply -f https://git.io/fwlG9
```

確認: kubectl get all -o wideでservice/nginx-svcのIPとポートを確認し、ブラウザでアクセスしてみる

Nginxのデプロイ

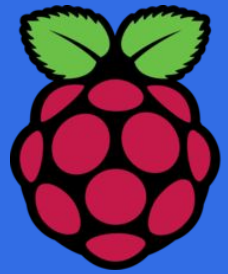
```
root@k8s-test:~# kubectl apply -f https://git.io/fwIG9
service/nginx-svc created
deployment.apps/nginx-dep created
root@k8s-test:~# kubectl get all -o wide
NAME                             READY   STATUS    RESTARTS   AGE     IP           NODE
pod/nginx-dep-57b696c78d-4kn8s   1/1     Running   0          6s      10.0.88.14   k8s-test
pod/nginx-dep-57b696c78d-6crbj   1/1     Running   0          6s      10.0.88.13   k8s-test

NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)           AGE     SELECTOR
service/kubernetes   ClusterIP   10.254.0.1    <none>        443/TCP           6d      <none>
service/nginx-svc    NodePort    10.0.78.252   <none>        10080:32239/TCP   6s      app=nginx

NAME                        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS   IMAGES        SELECTOR
deployment.apps/nginx-dep   2         2         2            2           6s      nginx        nginx:1.13    app=nginx

NAME                                   DESIRED   CURRENT   READY   AGE     CONTAINERS   IMAGES        SELECTOR
replicaset.apps/nginx-dep-57b696c78d   2         2         2       6s      nginx        nginx:1.13    app=nginx,pod-t
emplate-hash=1362527348
root@k8s-test:~#
```

kubernetes



Raspberry Pi

ここでは触れなかったことの一部

- 認証を有効にして、勝手にクラスタを操作されないようにする
- LoadBalancerやPersistentVolumesは環境ごとに実装が必要
  - MetalLB, Rookなども利用可能
- Raspberry Pi 3B+は有線LANがGbEに変更
- kube-dnsとかDashboardとか
- [9月以降に試すなら…] etcdctl v3.4以降、仕様変更の予定
  - ETCDCTL_API=2をつけることで、当分は従来のコマンドも提供
    - 例: etcdctl cluster-health → etcdctl endpoint health
    - etcdctl set xxx xxx → etcdctl put xxx xxx

References

- Kubernetes The Hard Way
- 3日間クッキング【Kubernetes のラズペリーパイ包み “サイバーエージェント風”】
- Creating a Custom Cluster from Scratch

Q&A