HITACHI
Inspire the Next

OSS Summit Tokyo 2017

Sharing knowledge and issues for applying Kubernetes® and Docker to enterprise system

6/1/2017

Natsuki Ogawa

Hitachi, Ltd.

HITACHI
Inspire the Next

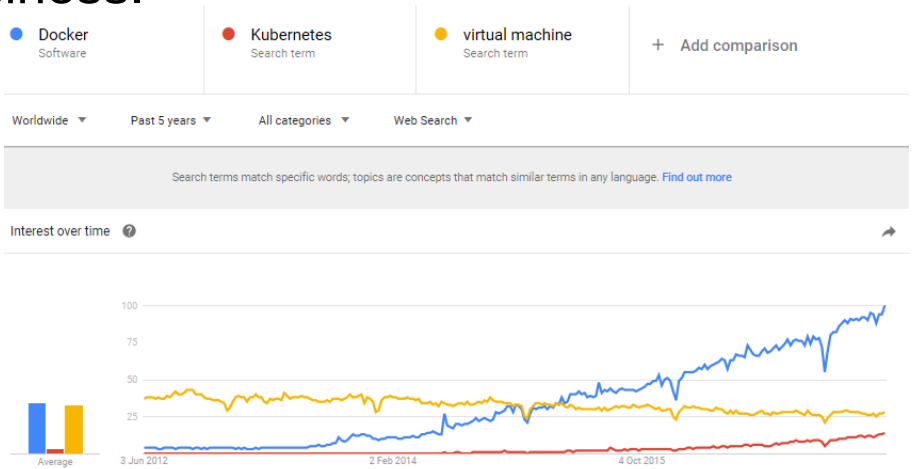
Contents

HITACHI
Inspire the Next

1.Background

1-1 Market trend

- Container technology is getting a lot of attention.
- This technology has been applied mainly to SoE systems.
- Companies whose most systems are SoR, such as financial companies, also start considering to apply it to their SoR systems, because they want to speed up their system development to focus on their new business.



Data source: Google Trends (www.google.com/trends)

1-2 Hitachi’s container service

HITACHI
Inspire the Next

- Hitachi has been providing container service to SoR system.
- For example, Hitachi has provided a financial company with its test environment on Kubernetes and Docker.

Through providing the system, we obtained knowledge and issues for applying them to enterprise system.

We will share them in this session.

1-3 Basic requirements for the system

HITACHI
Inspire the Next

Category	Details
Fast boot	Test environment must be able to be ready for auto tests quickly so that the tests can start soon to complete in a short time.
Grouping	Test applications must be grouped to execute system test on complicated structure.
Operating easily	For the beginners, the test operation such as start or stop machines must be executed easily.
Team development	Resource must be divided because the software are developed by hundred of developers and several teams or companies for each function.
Persistent Storage	Test evidence such as test data or log must be preserved.

1-4 Why Docker and Kubernetes for the system infrastructure

HITACHI
Inspire the Next

- Because the most important requirement is fast boot, Docker and Kubernetes are better than other traditional infrastructure such as virtual machine or bare metal.
- So, we adopted Docker and Kubernetes for the system.
- However, there is a problem that Dockerfile and Kubernetes manifests are difficult for beginners.

1-5 Implementation for basic requirement

HITACHI
Inspire the Next

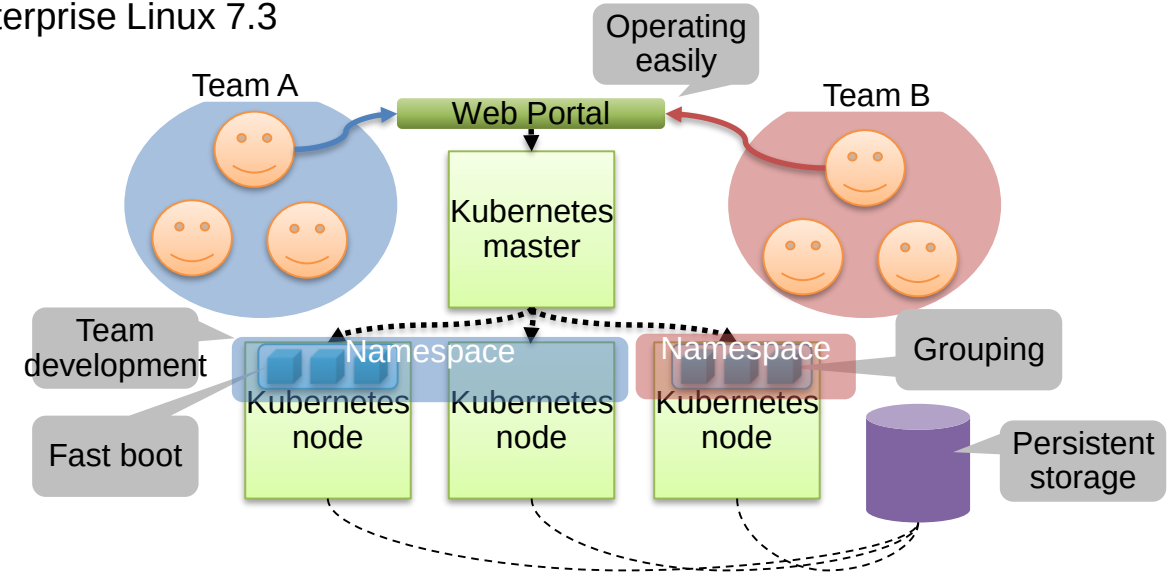
- The implementation for basic requirement.

Requirements	Implementation
Fast boot	Docker Container
Grouping	Kubernetes Pod
Operating easily	Web Portal
Team development	Kubernetes Namespace
Persistent Storage	Kubernetes HostPath and PersistentVolume

- For easy operation, we provide our web portal which wraps commands and files such as Dockerfile or Kubernetes manifests.

1-6 Overview of providing system

- Kubernetes version: 1.3.0
- Docker version: 1.10.3
- Red Hat Enterprise Linux 7.3



- Note that some problems in the chapter 2 and 3 happen only in above versions. In the latest versions, some of those problems may be resolved.

1-7 Problems for the system

HITACHI
Inspire the Next

- 3 problems came out after constructing the system.

Problems	Details
Problem A: Resource allocation to each team	A-1: Despite of development compliance, team resources are visible to other team.
	A-2: Hardware resources for each team is not allocated evenly.
Problem B: Middleware	B-1: Some kernel parameters cannot be configured on each container, independently.
	B-2: Some system call cannot be executed on a container.
Problem C: Storage	C-1: Kubernetes volume system depends on each cloud.
	C-2: Raw device cannot be mapped to container by function of Kubernetes volume.

HITACHI
Inspire the Next

2.Problems & Solutions

2 Problems & Solutions

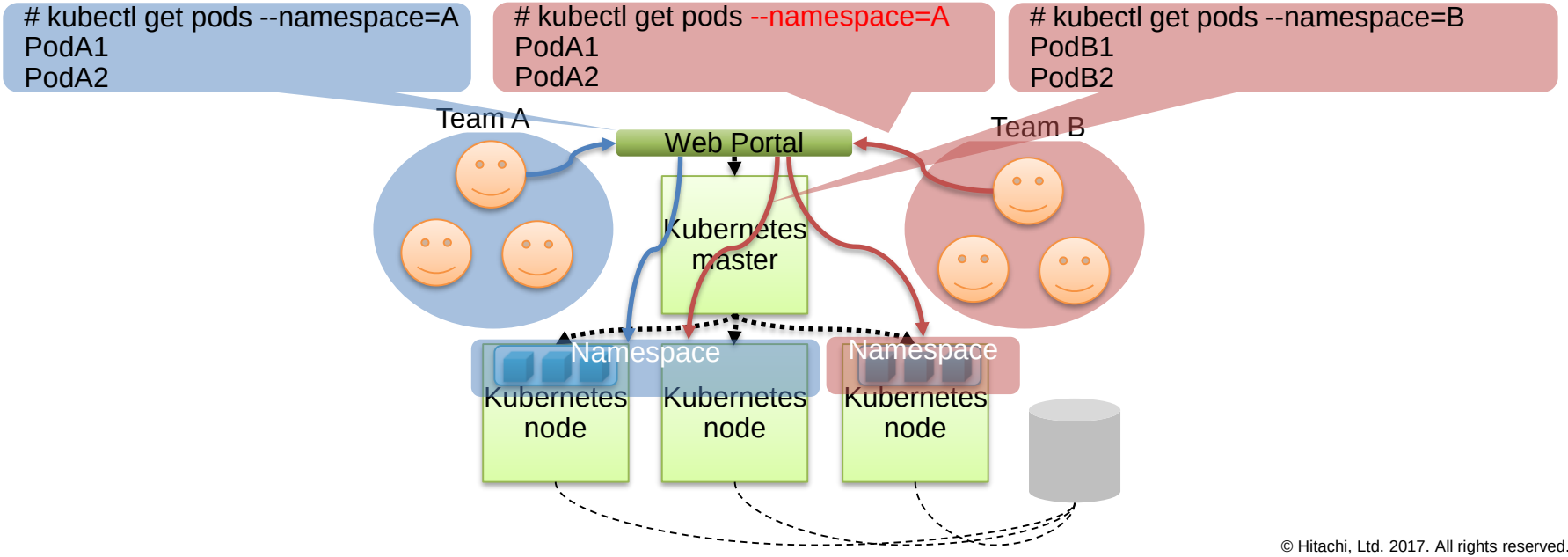
HITACHI
Inspire the Next

Problems	Details
Problem A: Resource allocation to each team	A-1: Despite of development compliance, team resources are visible to other team.
	A-2: Hardware resources for each team is not allocated evenly.
Problem B: Middleware	B-1: Some kernel parameters cannot be configured on each container, independently.
	B-2: Some system call cannot be executed on a container.
Problem C: Storage	C-1: Kubernetes volume system depends on each cloud.
	C-2: Raw device cannot be mapped to container by function of Kubernetes volume.

2- 1 Problem A-1：Current

HITACHI
Inspire the Next

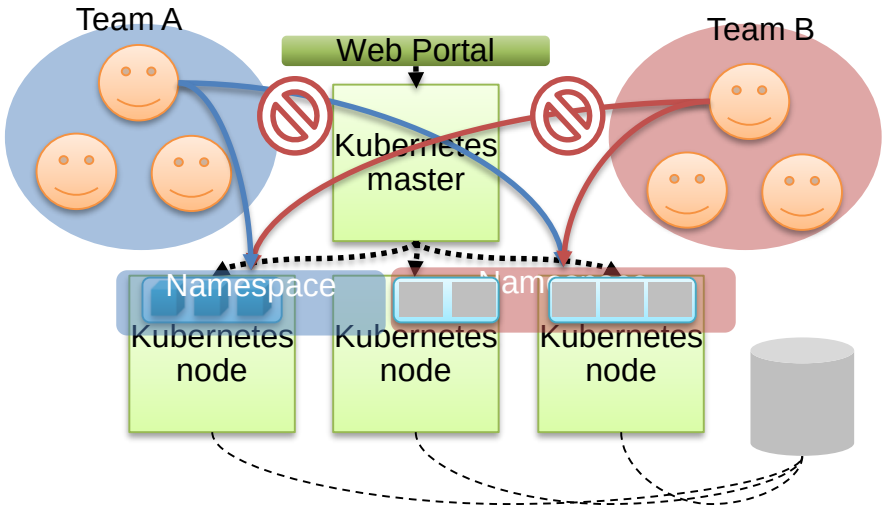
- Kubernetes can divide resources by using namespace.
- By allocating a different namespace for each team, resources can be divided for each team.
- K8s version 1.3 does not have a function to restrict access to a particular namespace, therefore, it can not prevent one team from accessing to the other team's resources.



2- 1 Problem A-1：Goal

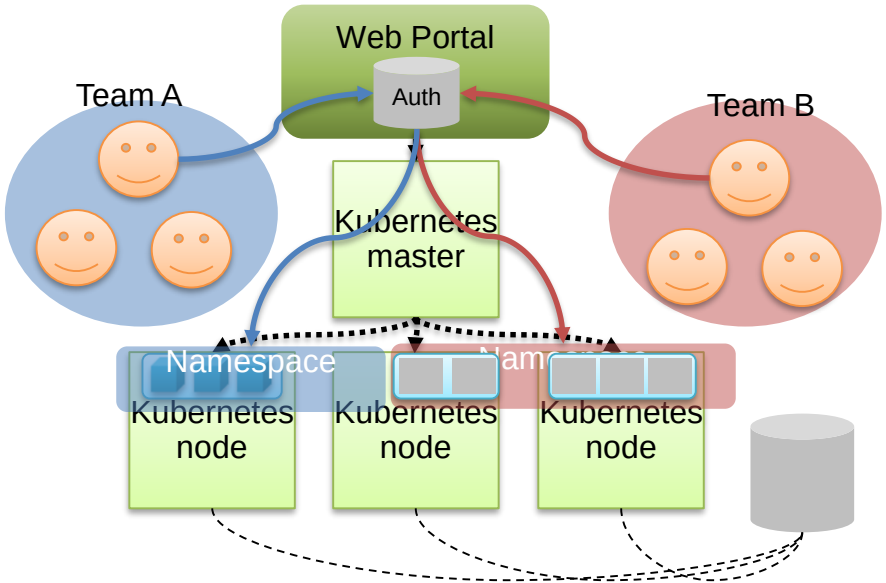
HITACHI
Inspire the Next

- Other team’s resources must not be able to see.



2- 1 Problem A-1: Solution

- Allow users access only via the portal, and make the portal show the user's team's resources only.
  - Portal is required to have an authentication function and user-to-team mapping information to identify users and to restrict access properly.



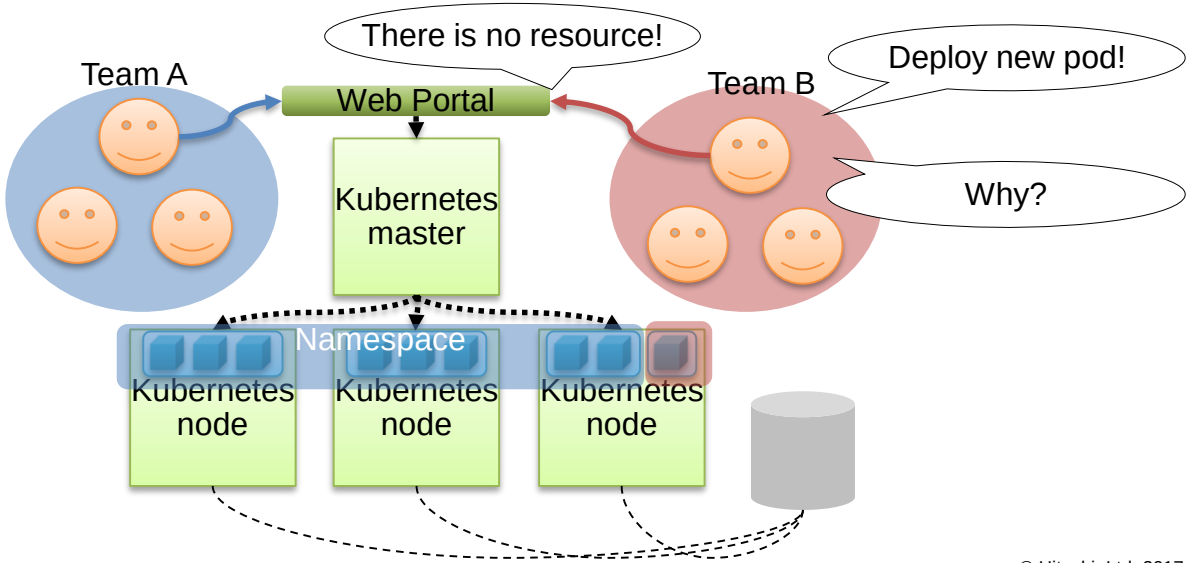
2 Problems & Solutions

HITACHI
Inspire the Next

Problems	Details
Problem A: Resource allocation to each team	A-1: Despite of development compliance, team resources are visible to other team.
	A-2: Hardware resources for each team is not allocated evenly.
Problem B: Middleware	B-1: Some kernel parameters cannot be configured on each container, independently.
	B-2: Some system call cannot be executed on a container.
Problem C: Storage	C-1: Kubernetes volume system depends on each cloud.
	C-2: Raw device cannot be mapped to container by function of Kubernetes volume.

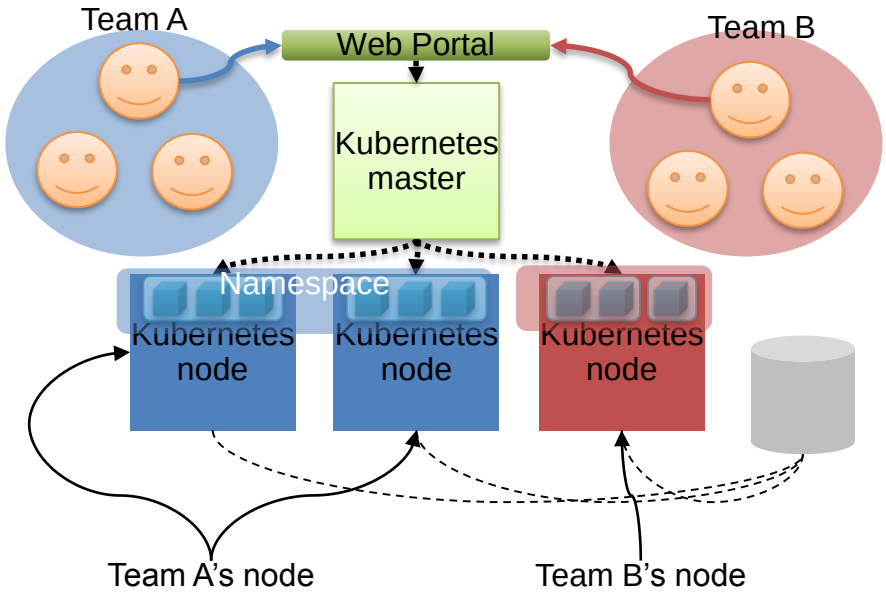
2- 1 Problems A-2: Current

- If all teams request resources freely, resources will be allocated unfairly to each team.
- For example, if Team A get resources a lot, Team B hardly can get resources.
- Hardware resource should be allocated fairly for each team.



2- 1 Problem A-2: Goal

- Resources for one team should not be exhausted by other teams' requests so that each team can always request a certain amount of resources as if it has dedicated nodes.
  - This should be achieved without increasing users' operation cost.



2- 1 Problem A-2：Solution

HITACHI
Inspire the Next

- There are 2 methods to solve this problem.
  - Quota for namespace
  - Restriction of placement destination using node label

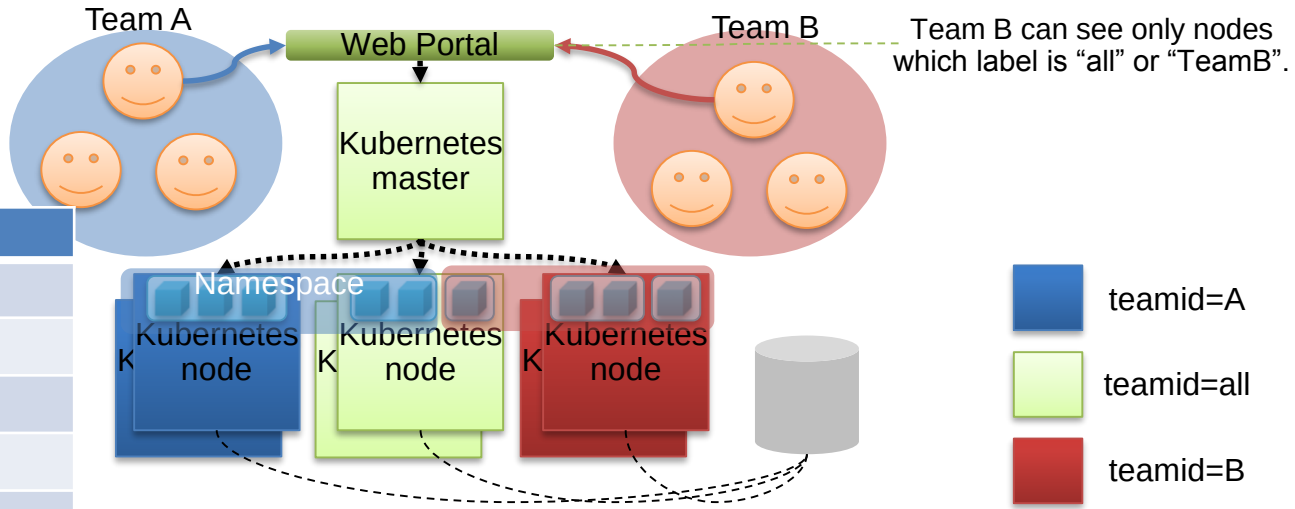
Methods	Pros	Cons
Quota	● Fine control of resources	● Administrator needs to manage Quota. ● User must set resource limit to each pod.
Node label	● Administrator needs to set labels only to nodes. ● Separate resources physically from other team's pods	● To make a pod deployed to the nodes with an intended node label, user must set the node label to each pod. ✓ Resolved in the portal ● Portal can assign the node label to the pod by using user-to-team mapping information instead of user operation.

✓ Adopted Node label suitable for this system

2- 1 Problem A-2：Solution

- In order to allocate the resources fairly to each team, we use dedicated nodes and shared nodes, which is identified with node labels.
  - The labels are managed by a system administrator.
- Each team can only use its own dedicated nodes and shared nodes via Portal.

Node	Label
#1	Team A
#2	Team A
#3	all
#4	all
#5	Team B
#6	Team B



2 Problems & Solutions

HITACHI
Inspire the Next

Problems	Details
Problem A: Resource allocation to each team	A-1: Despite of development compliance, team resources are visible to other team.
	A-2: Hardware resources for each team is not allocated evenly.
Problem B: Middleware	B-1: Some kernel parameters cannot be configured on each container, independently.
	B-2: Some system call cannot be executed on a container.
Problem C: Storage	C-1: Kubernetes volume system depends on each cloud.
	C-2: Raw device cannot be mapped to container by function of Kubernetes volume.

2- 2 Problem B-1：Current

HITACHI
Inspire the Next

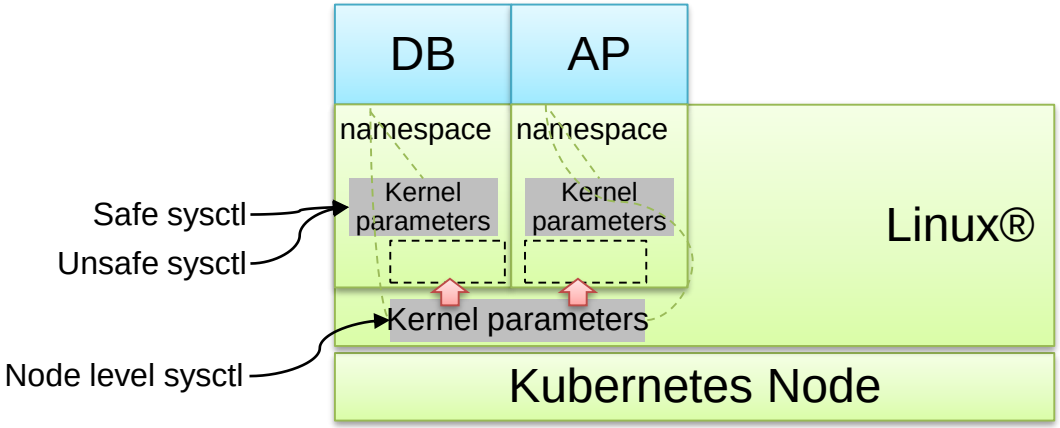
- There are 3 kinds of kernel parameters in the container.

Parameters	Detail	Range
Node level sysctl	It is set for each node and can not be set for each container.	Node
Unsafe.sysctl	Although it is set for each container, it may affect other containers	Pod
Safe.sysctl	It is set for each container, and it does not affect other container.	Pod

- We want to set kernel parameters to each container in order to use middle ware, for example DB.
- Setting Node level sysctls or unsafe.sysctls will affect another container.

2- 2 Problem B-1: Goal

- Kernel parameters can be set without affecting other containers.
  - For this purpose, kernel parameters classified as Node level / unsafe should be able to also be set without affecting other containers.



2- 2 Problem B-1：Solution

HITACHI
Inspire the Next

- We place Pods as following rules.

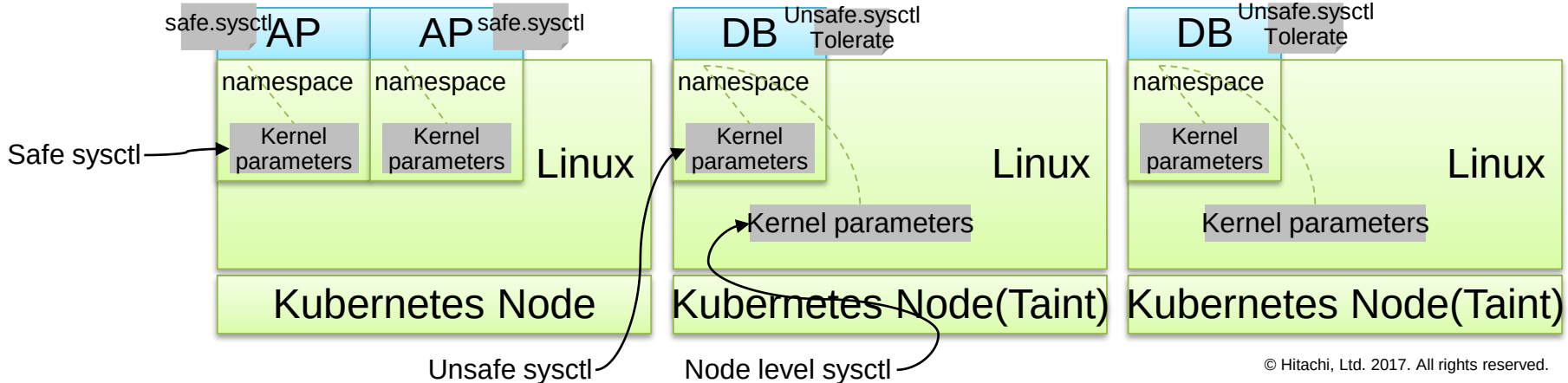
Pod sysctl settings	Condition	Destination node
Only safe.sysctl is set	N/A	Shared
Unsafe.sysctl or Node level sysctl is set	Any values of parameter of Pod are different from any existent Pods.	Dedicated
	All values of parameter of Pod are the same as any existent Pods.	The node which has the same parameters.

2- 2 Problem B-1: Solution (cont'd)

HITACHI
Inspire the Next

The rules can be achieved by using Tolerate and Taint features of Kubernetes.

- Pods with only safe.sysctl:
  1. User deploys Pod without Tolerate.
  2. Kubernetes places the Pod to a node without taint.
- Pods with unsafe.sysctl or Node level sysctl:
  1. User sets sysctl parameters to the node if using Node level sysctl.
  2. User adds Taint for the parameters to the node.
  3. User deploys Pod with Tolerate corresponding to the Taint.
  4. Kubernetes places the Pod to the node with Taint corresponding to the specified Tolerate.



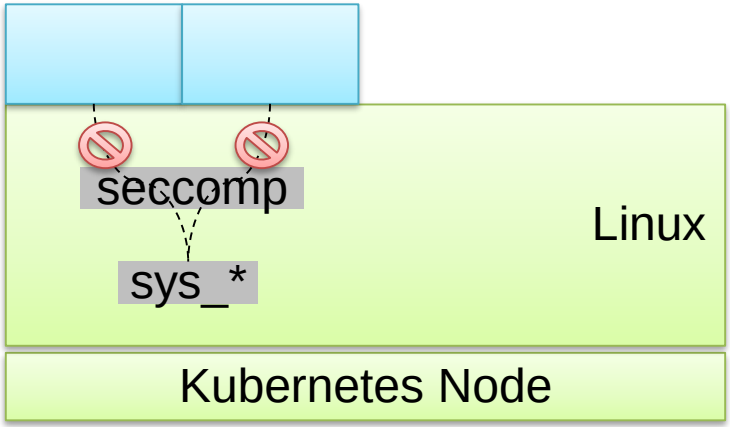
2 Problems & Solutions

HITACHI
Inspire the Next

Problems	Details
Problem A: Resource allocation to each team	A-1: Despite of development compliance, team resources are visible to other team.
	A-2: Hardware resources for each team is not allocated evenly.
Problem B: Middleware	B-1: Some kernel parameters cannot be configured on each container, independently.
	B-2: Some system call cannot be executed on a container.
Problem C: Storage	C-1: Kubernetes volume system depends on each cloud.
	C-2: Raw device cannot be mapped to container by function of Kubernetes volume.

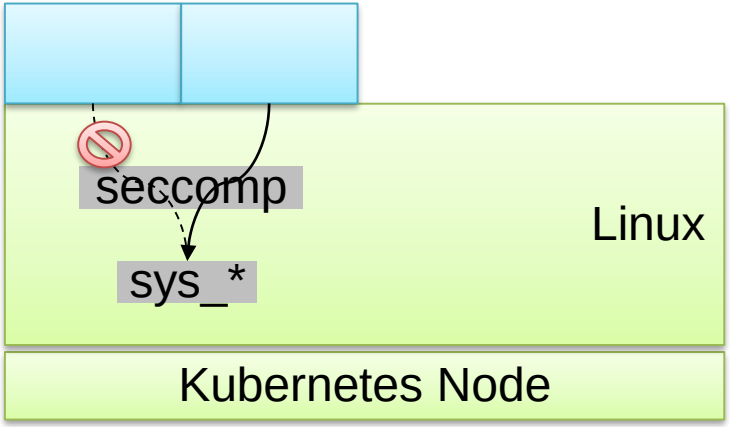
2- 2 Problem B-2：Current

- Some system calls are restricted by default by Docker.
  - Application’s operation is restricted.
    - Ex. core dump



2- 2 Problem B-2：Goal

- System calls can be executed within a container.
- Above settings are enable for each container.
- However, extra permissions are not desirable due to security problems.



2- 2 Problem B-2：Solution

HITACHI
Inspire the Next

- To make system call executable, it is necessary to set the seccomp=unconfined and either of the following settings.
  - cap_add： Add specified Linux capabilities
  - privileged：Add all Linux capabilities

✓ We adopted cap_add because it can restrict capabilities.

- Ex. Database server's Kubernetes manifest

```
seccomp.security.alpha.kubernetes.io/pod: unconfined
securityContext:
  capabilities:
    add:
      - NET_ADMIN
      - SYS_RESOURCE
      - LEASE
      - IPC_LOCK
      - IPC_OWNER
```

2 Problems & Solutions

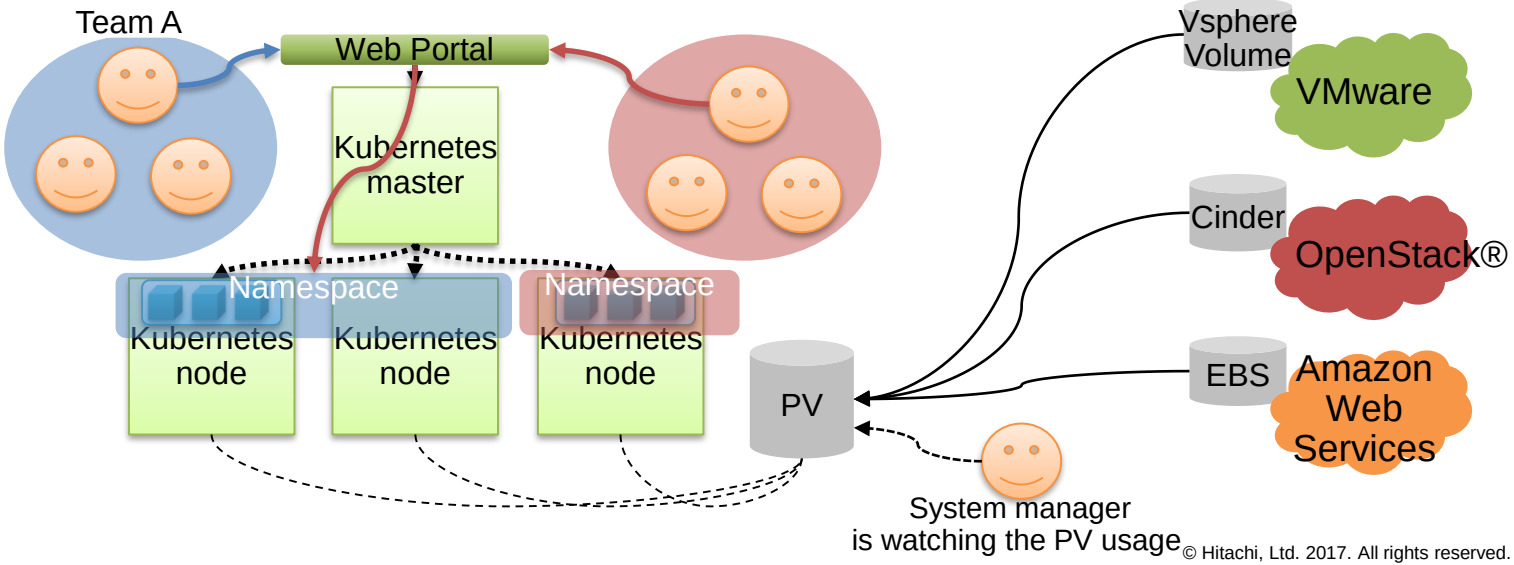
HITACHI
Inspire the Next

Problems	Details
Problem A: Resource allocation to each team	A-1: Despite of development compliance, team resources are visible to other team.
	A-2: Hardware resources for each team is not allocated evenly.
Problem B: Middleware	B-1: Some kernel parameters cannot be configured on each container, independently.
	B-2: Some system call cannot be executed on a container.
Problem C: Storage	C-1: Kubernetes volume system depends on each cloud.
	C-2: Raw device cannot be mapped to container by function of Kubernetes volume.

2- 3 Problem C-1: Current

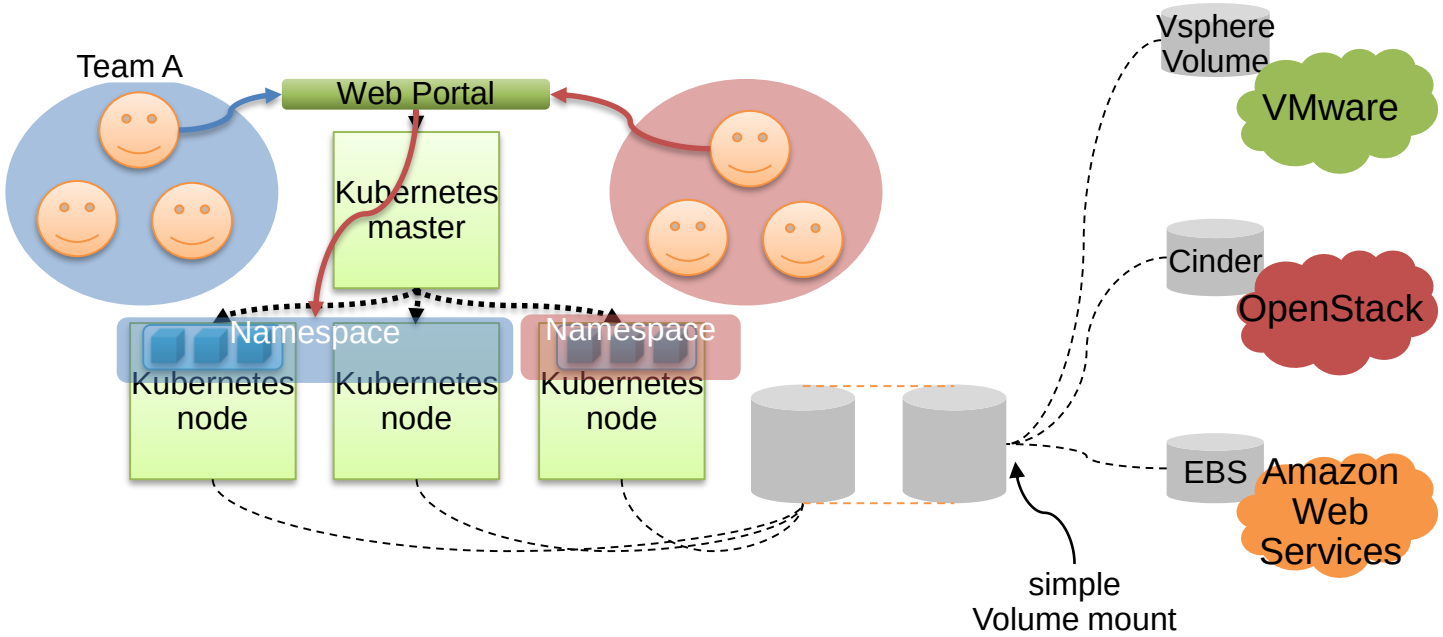
HITACHI
Inspire the Next

- Kubernetes has Persistent volume (PV) which can preserve data persistently on Pod.
  - AWS EBS, OpenStack Cinder and other storage can be used as a resource of PV.
  - A system administrator must prepare some PV before users use them.
  - User specifies the desired size as a Persistent Volume Claim (PVC) and Kubernetes assigns a part of the PV to Pod as PVC.
- The problem of PV and PVC
  - Administrator must manage PV, user must manage PVC.
  - In the hybrid cloud, since it is necessary to prepare PV from each cloud, management cost increase.



2- 3 Problem C-1: Goal

- Test data and log are preserved permanently by the same way across the clouds.
- Administrator’s operation should be reduced more.



2- 3 Problem C-1：Solution

HITACHI
Inspire the Next

- We consider whether it can be solved by using NFS independent from clouds.

Methods	Pros	Cons
NFS + HostPath	● There is no need to consider management of association between containers and volumes on the system side.	● Low I/O performance ✓ However, this system needs less data capacity and not high I/O performance

✓ NFS + HostPath was adopted for this system.

2 Problems & Solutions

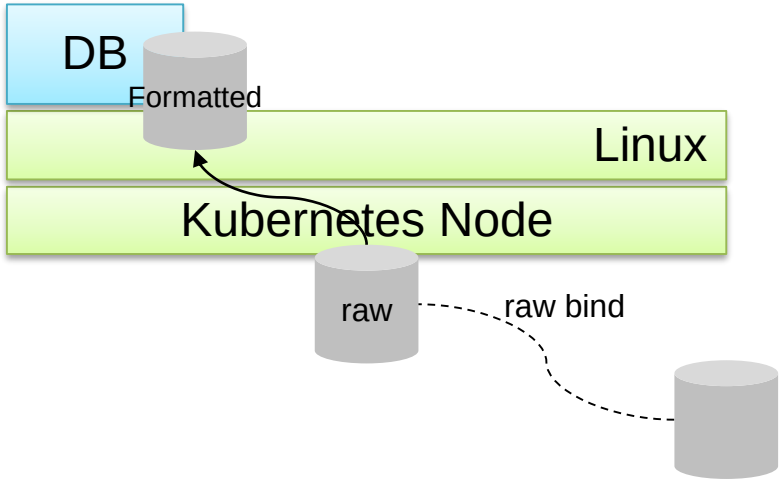
HITACHI
Inspire the Next

Problems	Details
Problem A: Resource allocation to each team	A-1: Despite of development compliance, team resources are visible to other team.
	A-2: Hardware resources for each team is not allocated evenly.
Problem B: Middleware	B-1: Some kernel parameters cannot be configured on each container, independently.
	B-2: Some system call cannot be executed on a container.
Problem C: Storage	C-1: Kubernetes volume system depends on each cloud.
	C-2: Raw device cannot be mapped to container by function of Kubernetes volume.

2- 3 Problem C-2: Current

HITACHI
Inspire the Next

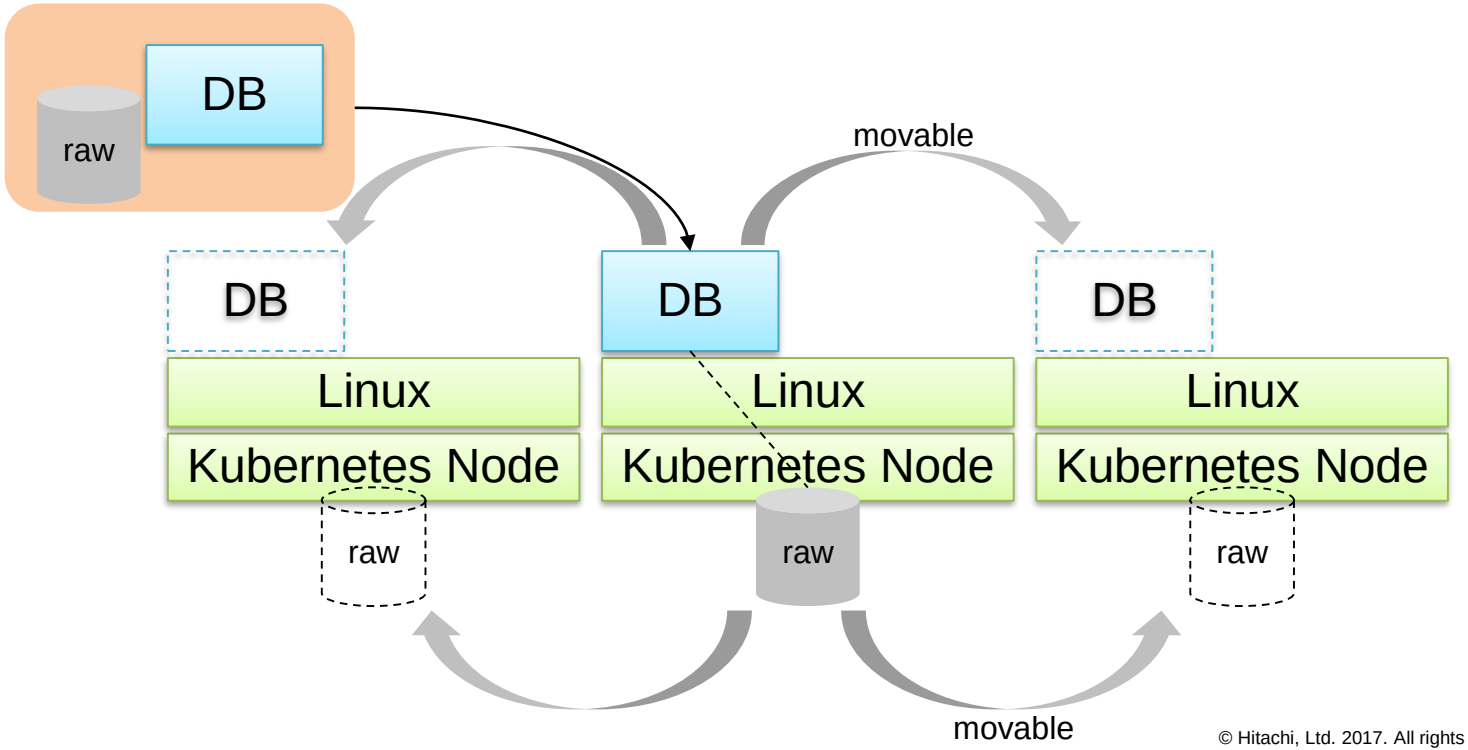
- NFS + HostPath are used for the DataBase area.
- Raw device improves performance of DataBase etc.
- However, NFS + HostPath cannot be used for raw device on a host.
  - Even if we use PV instead of NFS + HostPath, the volume mapped in the container is formatted as a file system and raw devices can not be used.



2- 3 Problem C-2: Goal

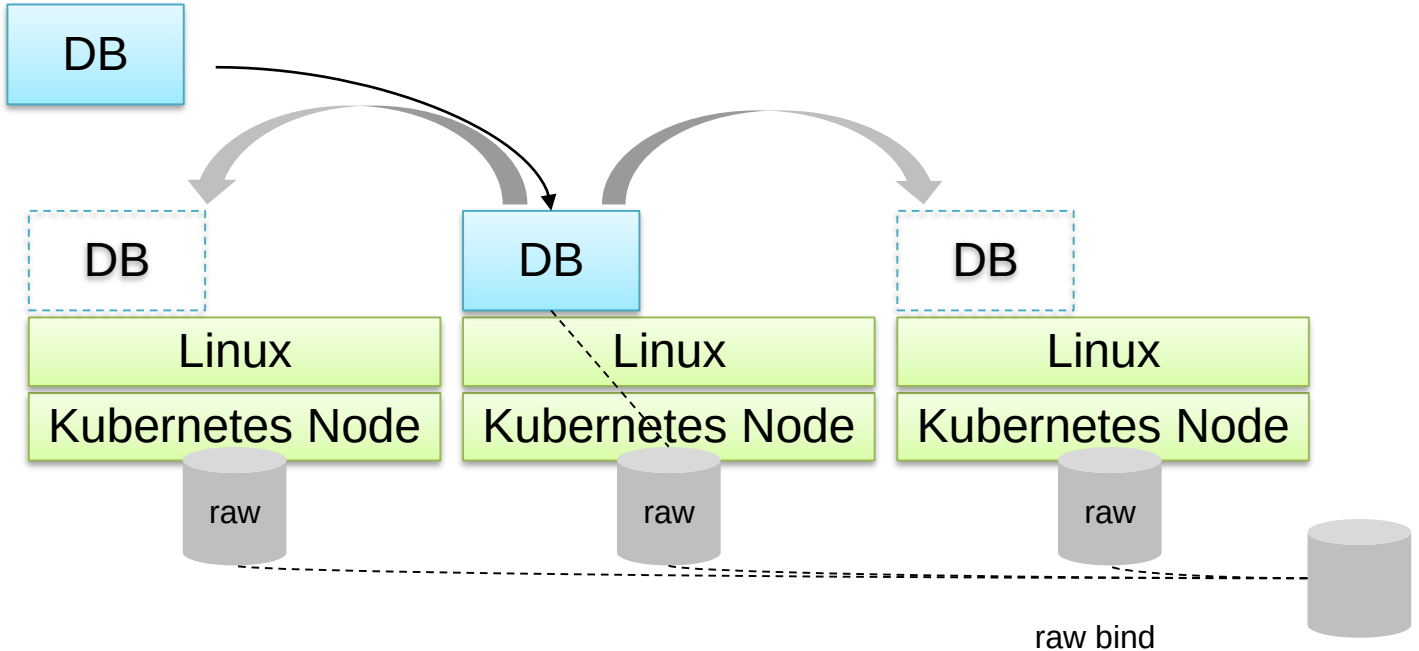
HITACHI
Inspire the Next

- Raw device can be used in a container.
- Raw device should be accessible from the container, even when the container is moved to different node.



2- 3 Problem C-2：Solution

- Create and connect raw devices to the Kubernetes nodes which can be used for DB.
- Create containers which needs raw device with the following settings.
  - Grant privileges for host mapping to the Pod to connect raw device.
  - Connect raw device with HostPath.



HITACHI
Inspire the Next

3.Evaluation

3 Evaluation

HITACHI
Inspire the Next

Problems	Details	Feasibility
Problem A: Resource allocation to each team	A-1: Despite of development compliance, a team resource is seen from other team.	✓ Realized by Namespace, Web Portal
	A-2: The hardware resource for each team is not allocated evenly.	✓ Realized by Node label and Web Portal
Problem B: Middleware	B-1: Some kernel parameters cannot be configured on a container.	✓ Realized by Node taint
	B-2: Some system call cannot be executed on a container.	✓ Realized by seccomp, cap_add
Problem C: Storage	C-1: Kubernetes volume system depends on each cloud.	✓ Realized by NFS and HostPath
	C-2: Raw device cannot be mapped to container by function of Kubernetes volume.	✓ Realized by host device mount

3 Evaluation

HITACHI
Inspire the Next

Problems	Details	The room of improvement	Ideal Kubernetes function
Problem A: Resource allocation to each team	A-1: Despite of development compliance, a team resource is seen from other team.	The current implementation is complicated, but Kubernetes Role-Based Access Control may make the implementation easier.	N/A
	A-2: The hardware resource for each team is not allocated evenly.	N/A	N/A
Problem B: Middleware	B-1: Some kernel parameters cannot be configured on a container.	Pod scheduling preparation takes time because user checks and set for it manually.	Pod scheduling by sysctl settings
	B-2: Some system call cannot be executed on a container.	N/A	N/A
Problem C: Storage	C-1: Kubernetes volume system depends on each cloud.	I/O performance should be improved because of using NFS.	N/A
	C-2: Raw device cannot be mapped to container by function of Kubernetes volume.	It is necessary to connect the device on all nodes manually because there are no function to control raw device.	Raw device connect automatically on Pod.

3 Evaluation (cont'd)

HITACHI
Inspire the Next

- We provided Kubernetes and Docker to the testing environment of SoR environment.

  ✓ Improve development test speed.

- How quickly did the test environment build?
  - Traditional environment: Copy VM, manual configuration.
  - Current environment: Launching Pods from Web Portal.
  - Assumed that 3 VM was aggregated into one pod.

  ✓ The time for building test environment improved 20 times faster than that for traditional environment.

HITACHI
Inspire the Next

4.Remaining issues

4-1 Other Knowledge and issues of Kubernetes and Docker

HITACHI
Inspire the Next

- Pods collaboration
  - There is no official way to control the order of boot sequence between Pods. We found 3 workarounds. We hope Kuberetes provides official way.
    - Init container
    - 3rd party plugin.
      - https://github.com/Mirantis/k8s-AppController
    - ReadinessProbe & command and shell script.
- For legacy application
  - In Kubernetes, Docker's --add-host can not be used.
  - Applications referring to /etc/hosts do not work at startup.
- Docker compatibility
  - When migrating from Docker 1.9 to Docker 1.10, we must change default value of seccomp. If you need all privileges, Unconfined may be required.
    - Some functions of the application that was running up to Docker 1.9 will stop working.
    - Validation required for Docker update.

4-2 knowledge from the system

HITACHI
Inspire the Next

- Docker registry's CLI is poor, it cannot delete images.
  - There are various third party products, but the deletion function, the authentication function, etc. are inadequate.
- Writing Dockerfile and Kubernetes manifest is difficult for beginners.
  - There should be a easy generation tool such as using GUI.
- Troubleshoot (log) problem
  - There is too much log output of the Kubedns container.
    - Resolved https://github.com/kubernetes/kubernetes/pull/36013

HITACHI
Inspire the Next

5.Summary

5 Summary

HITACHI
Inspire the Next

- We implemented the test system environment for a financial company with Docker and Kubernetes.
- It speeded up development without changing development process.
- We shared knowledge and issues for Kubernetes and Docker to enterprise system.

5 Summary

HITACHI
Inspire the Next

- Since Kubernetes is under development, functions for enterprise system are being implemented one after another. We recommend to catch up the latest information.
- Due to the difference in the combination of Kubernetes and Docker versions, compatibility problems tend to occur, so it should be applied to production after testing. We recommend to test the combination again when updating.
- Dockerfile and Kubernetes manifest are still difficult for end users to use directly. GUI or other mechanisms are required for them to use easily.

Trademarks

HITACHI
Inspire the Next

- HITACHI is a registered trademark of Hitachi, Ltd.
- Red Hat is a trademark or a registered trademark of Red Hat Inc. in the United States and other countries.
- Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.
- Docker and the Docker logo are trademarks or registered trademarks of Docker, Inc. in the United States and/or other countries. Docker, Inc. and other parties may also have trademark rights in other terms used herein.
- Kubernetes® is a registered trademark of The Linux Foundation.
- The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.
- Amazon Web Services is a trademark of Amazon.com, Inc. or its affiliates in the United States and/or other countries.
- Other company and product names mentioned in this document may be the trademarks of their respective owners.

HITACHI
Inspire the Next

END

Sharing knowledge and issues for applying Kubernetes and Docker to enterprise system

6/1/2017

Natsuki Ogawa

Hitachi, Ltd.

HITACHI

Inspire the Next