Multi-Master PostgreSQL Cluster on Kubernetes

PGConf.Asia 2017

NTT OSSセンタ Masanori Oyama/大山真実

Who Am I

Masanori Oyama / 大山 真実

twitter @ooyamams1987

slideshare https://www.slideshare.net/ooyamams

Work

- PostgreSQL engineering support and consultation.
- Extensive quality verification of PostgreSQL releases.
- PostgreSQL Security
  - ➢ PostgreSQL Security. How Do We Think? at PGCon 2017 https://www.slideshare.net/ooyamams/postgresql-security-how-do-we-think-at-pgcon-2017

Prev work

- Hadoop engineering support and consultation at NTT DATA Corp.
  - ➢ I had managed a big Hadoop cluster (1000 node, 200PB!).

Today's Topic

I will talk about …

- Introduce Next Generation Multi Master PostgreSQL Cluster and share some details about a PoC of IoT log platform.

- Share the knowledge of PostgreSQL cluster management using Kubernetes.

Today’s Topic

I will talk about ...

Please read the today's morning presentation material "Built-in Sharding“!

- ~~Introduce Next Generation Multi Master PostgreSQL Cluster and share some details about a PoC of IoT log platform.~~

- Share the knowledge of PostgreSQL cluster management using Kubernetes.

Today’s Topic

I will talk about …

Please read the today's morning presentation material "Built-in Sharding“!

- ~~Introduce Next Generation Multi Master PostgreSQL Cluster and share some details about a PoC of IoT log platform.~~

- Share the knowledge of PostgreSQL cluster management using Kubernetes.

I will focus on this today.

Table of Contents

1. Introduction
2. Crunchy PostgreSQL Containers on OpenShift
3. Multi-Master PostgreSQL Cluster on OpenShift
4. Conclusion

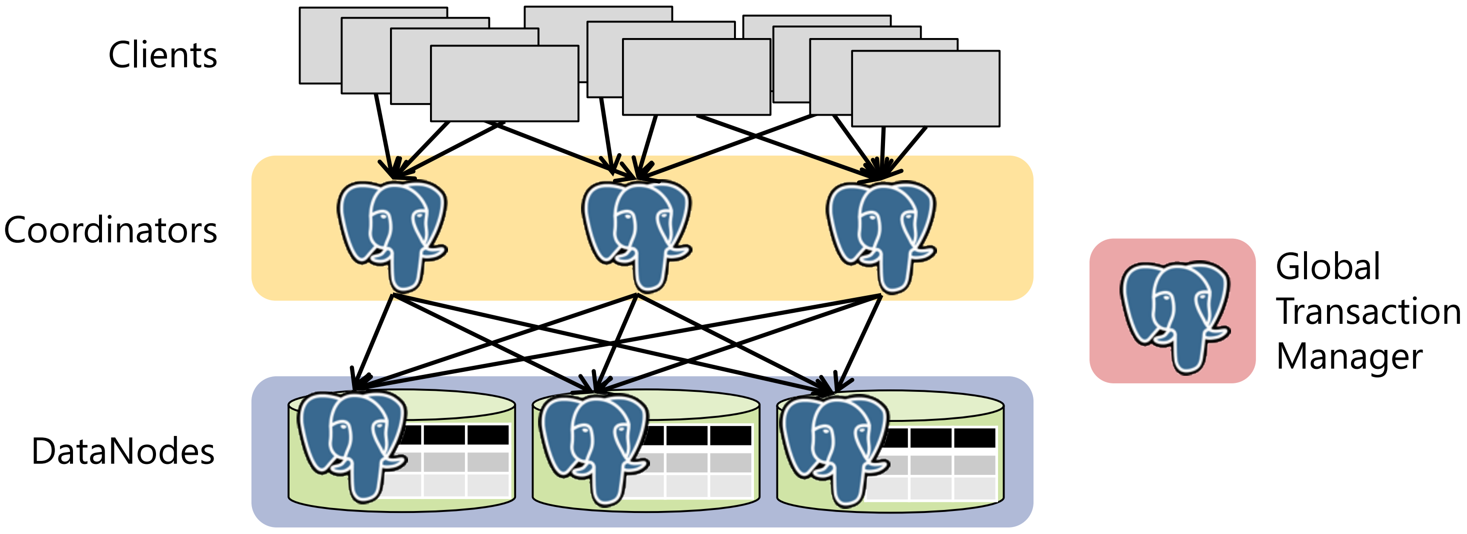
Table of Contents

1. **Introduction**
2. Crunchy PostgreSQL Containers on OpenShift
3. Multi-Master PostgreSQL Cluster on OpenShift
4. Conclusion

1. Introduction

Postgres-XC

- Multi-Master PostgreSQL Cluster forked from PostgreSQL by NTT
- Scale out for Read/Write workloads
- The development ended in 2014



Now, We are contributing back this experiences to the PostgreSQL core.
Please see "PostgreSQL Built-in Sharding" of today's morning presentation.

1. Introduction

Why scale out?

It is hard to improve write performance by scale up (by purchasing high-performance hardware).

Major bottleneck points of PostgreSQL

- Get transaction snapshot
- Write WAL

There is a limit of performance improvement even if you use many core server and NVMe.

To solve these bottlenecks, a drastic improvement is necessary.

- New Transaction Mechanism for many core machine
- Parallel WAL Mechanism or New WAL Writer Mechanism for NVMe

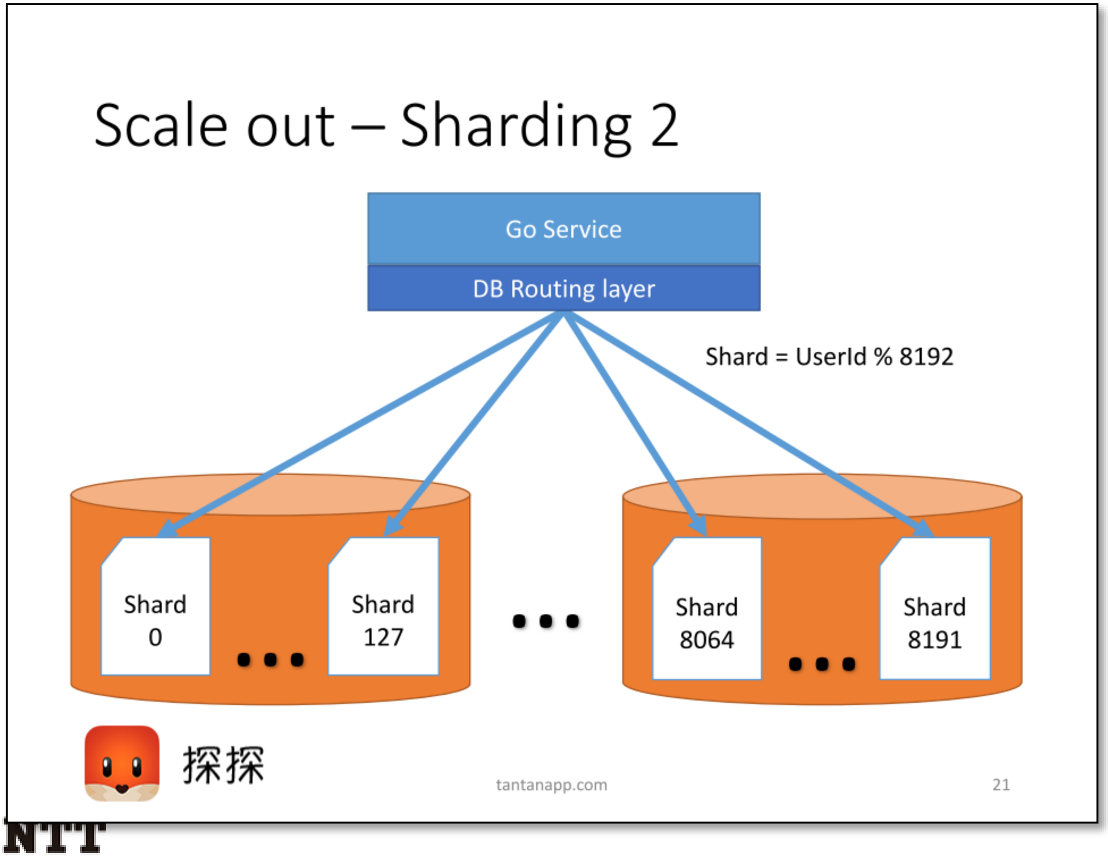
Scale out of PostgreSQL is one of the workarounds to avoid these bottlenecks.

1． Introduction

“Instagram” Way

Tantan told about the homemade version similar to the “Instagram” way at last year PGconf.Asia.

http://www.pgconf.asia/JP/wp-content/uploads/2016/12/From-0-to-350bn-rows-in-2-years-PgConfAsia2016-v1.pdf



Scale out – Sharding 2

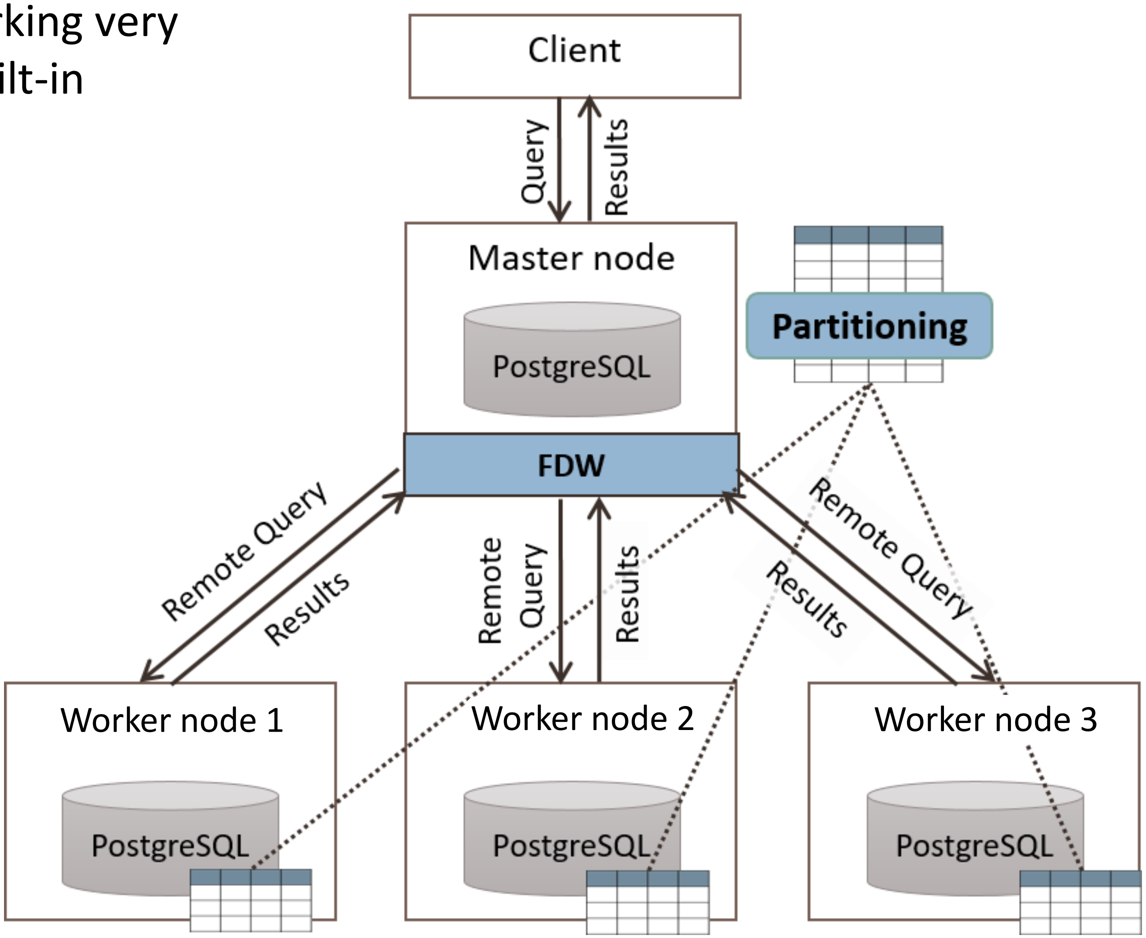
- 22TB / 350bn rows Biggest table.
- 270k tuple writes / sec.
- 1.3M TPS Aggregated over all databases

Applications need to control which databases are accessed and satisfy the ACID properties with transaction.

1． Introduction

PostgreSQL Built-in Sharding

PostgreSQL community is working very hard to realize PostgreSQL Built-in Sharding.



1. Introduction

Even if we could get a *Perfect PostgreSQL Cluster*, we have to spend a lot of time to manage the cluster.

For example,

- Orchestration
- HA and Recovery
- Backup
- Monitoring
- Security
- Resource Management
  - Load Balancing
  - Scale out (Add/Delete Coordinator/Shard)

Distributed systems are hard!

1． Introduction

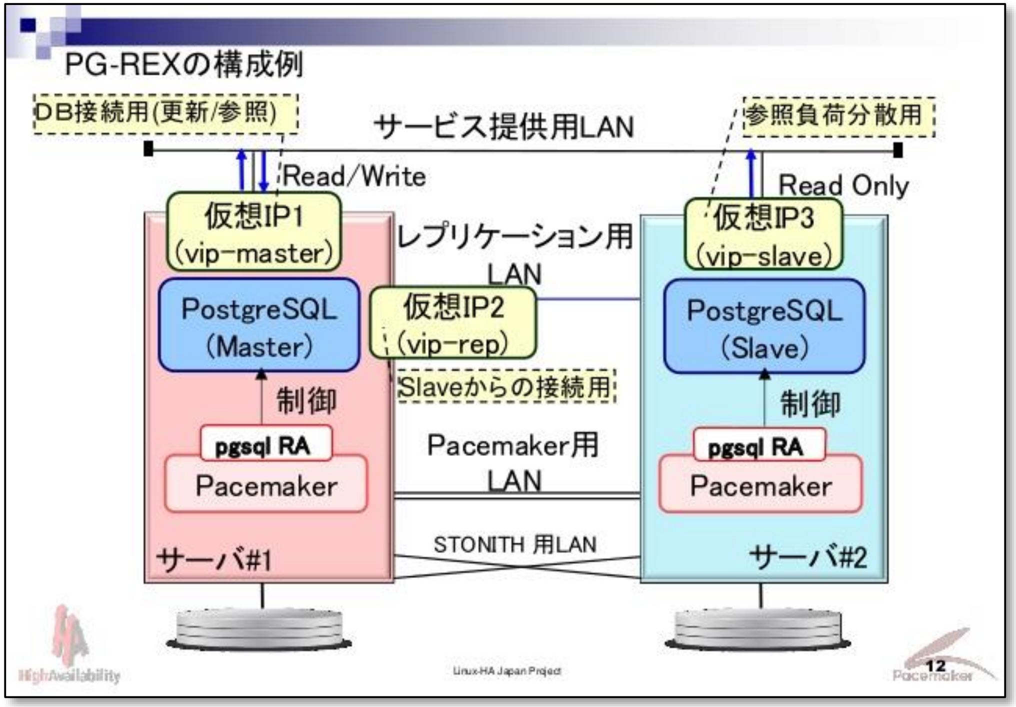
The Difficulty of Scale Out

For example, about High Availability.

With a ordinary PostgreSQL, we have to use a middleware to achieve HA.

PG-REX

PostgreSQL HA solution with Pacemaker



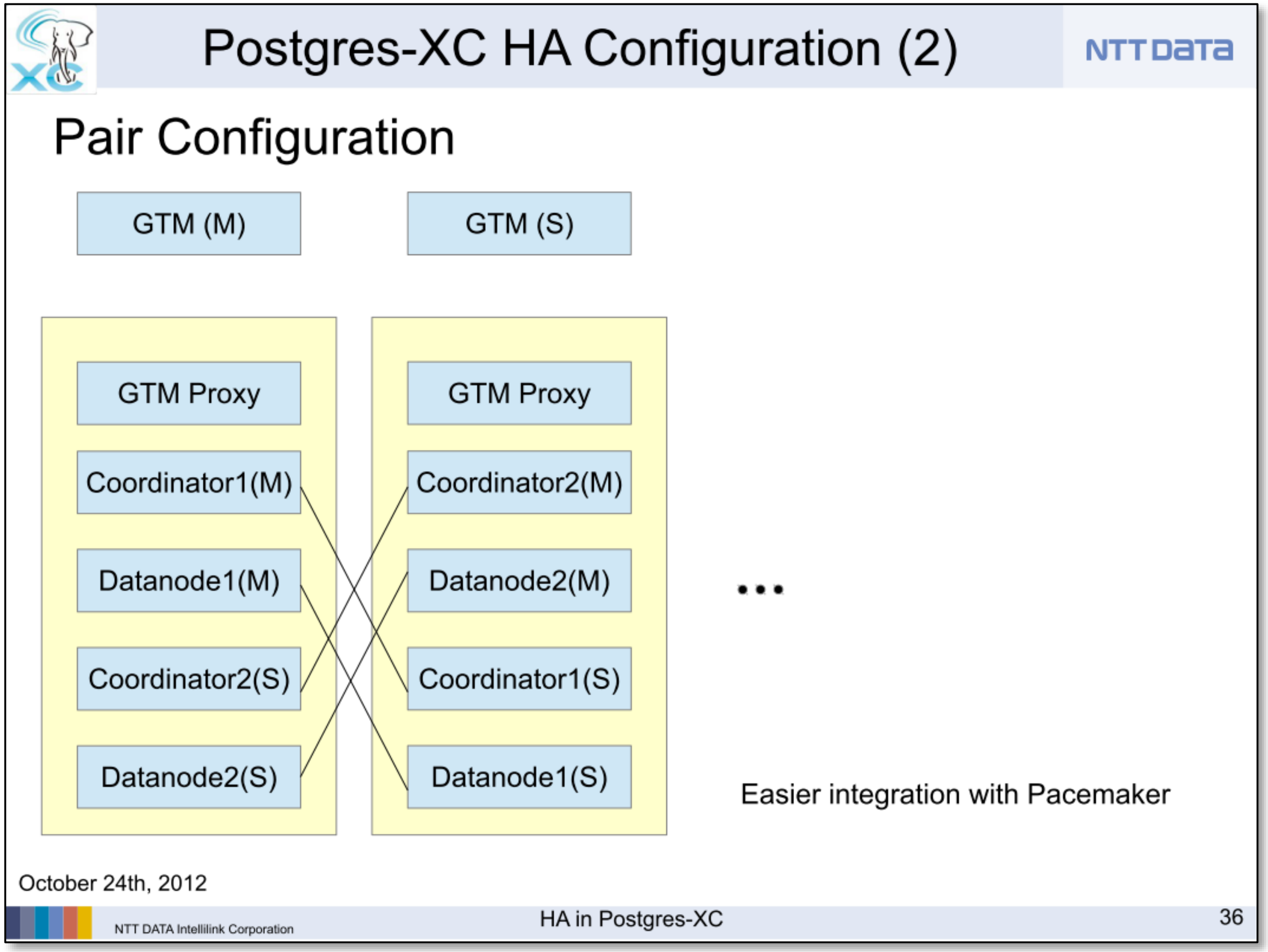
PG-REXの構成例

https://www.slideshare.net/kazuhcurry/pgrexpacemaker (Japanese)

How about the PostgreSQL cluster?

1． Introduction

HA of PostgreSQL-XC



Postgres-XC HA Configuration (2)

NTT DaTa

Pair Configuration

Easier integration with Pacemaker

October 24th, 2012

NTT DATA Intellilink Corporation

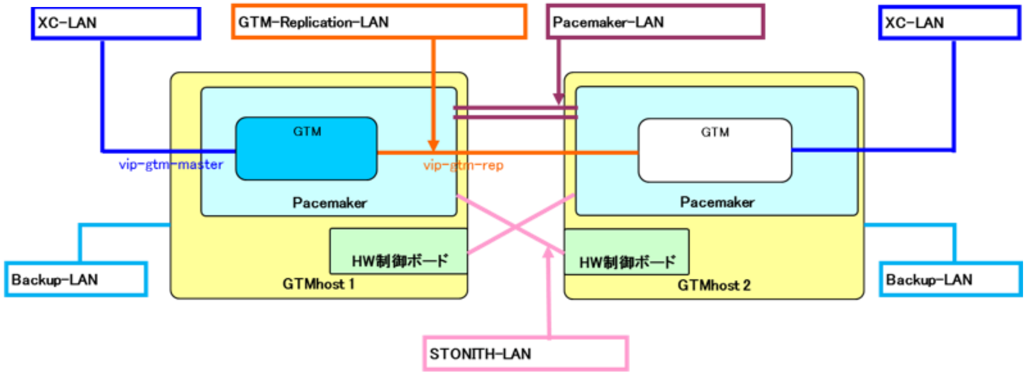
HA in Postgres-XC

36

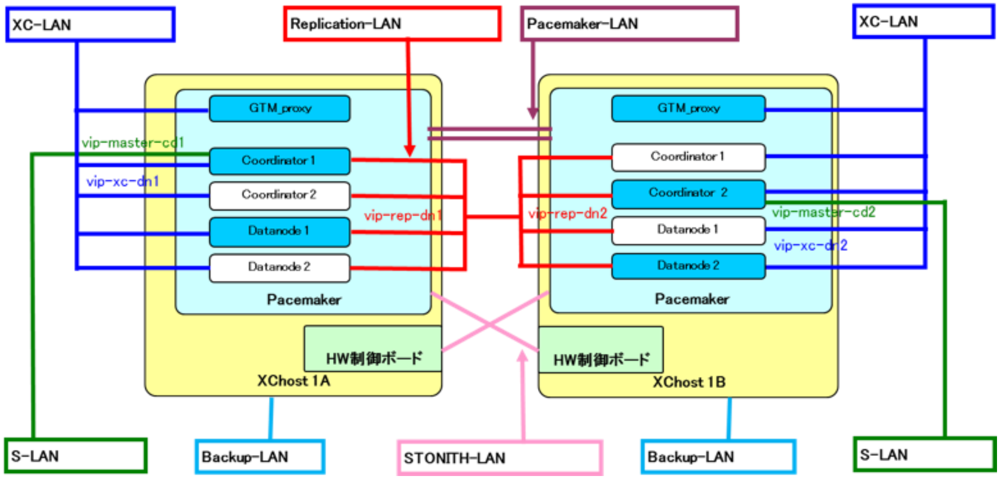
https://wiki.postgresql.org/images/4/44/Pgxc_HA_20121024.pdf

HA of PostgreSQL-XC

GTM Server



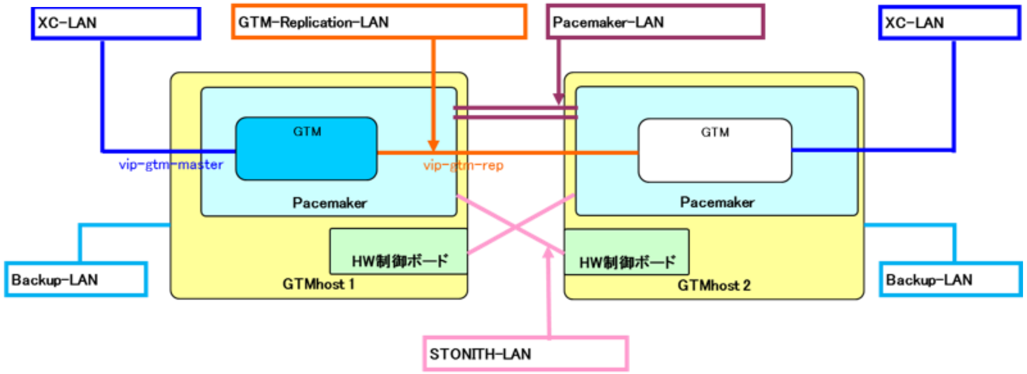
Coordinator and DataNode Servers



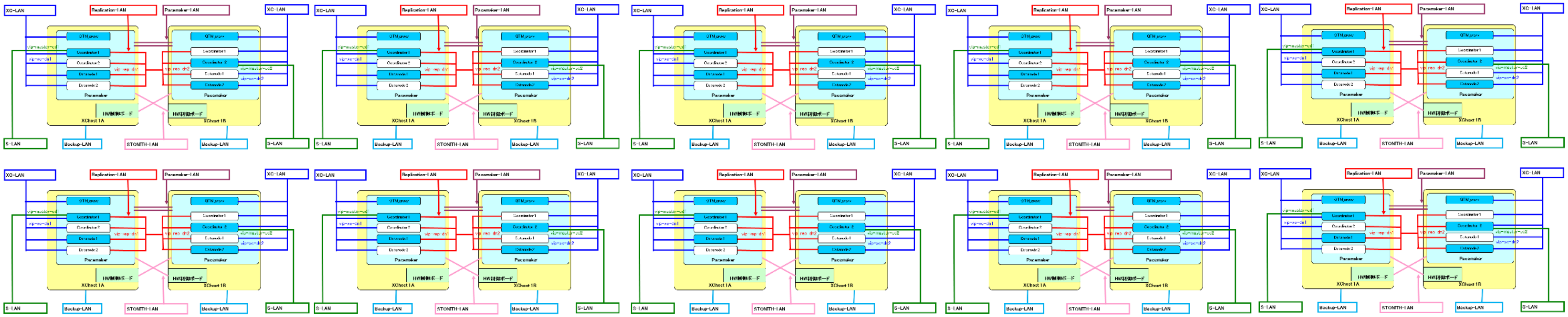
HA of PostgreSQL-XC 10 shard

How many HA settings are needed!?

GTM Server



Coordinator and DataNode Servers

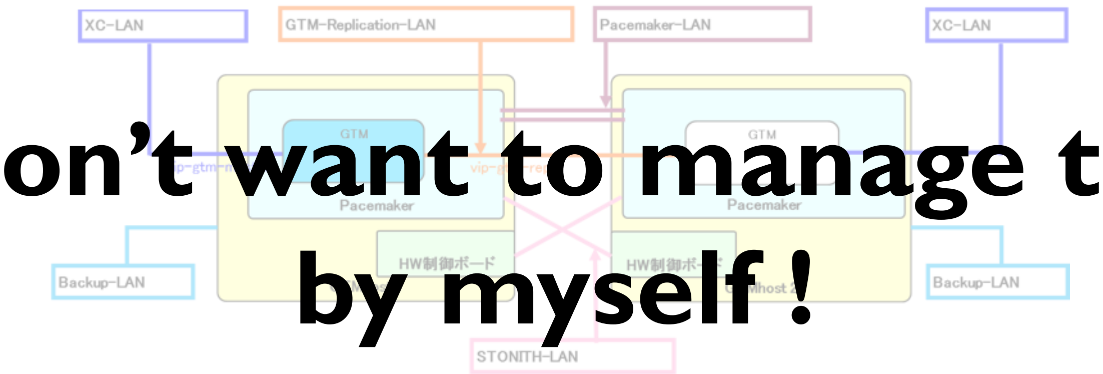


1． Introduction

HA of PostgreSQL-XC 10 shard

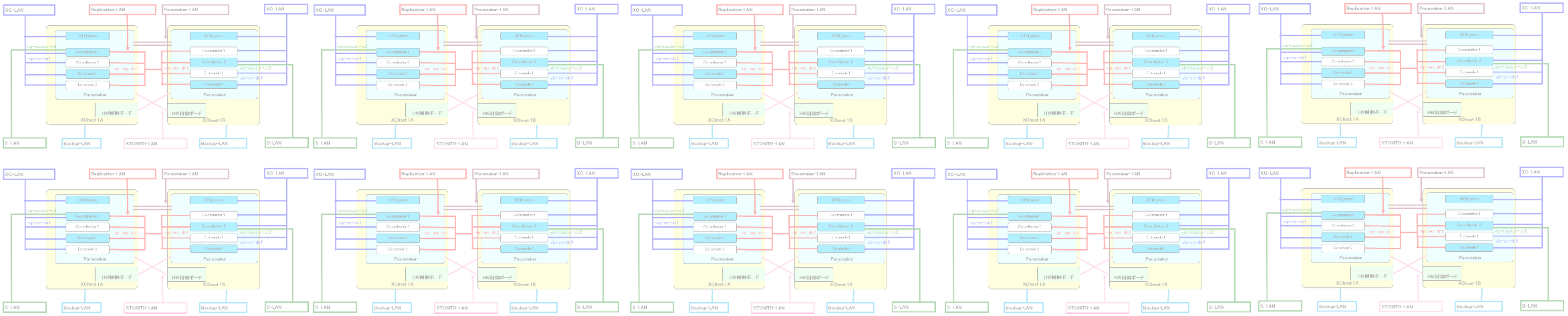
How many HA settings are needed!?

GTM Server



I don’t want to manage this by myself !

Coordinator and DataNode Servers



1. Introduction

How do people solve this?

PostgreSQL Related Slides and Presentations

https://wiki.postgresql.org/wiki/PostgreSQL_Related_Slides_and_Presentations

PGCONF.EU 2017

- USING KUBERNETES, DOCKER, AND HELM TO DEPLOY ON-DEMAND POSTGRESQL STREAMING REPLICAS
  https://www.postgresql.eu/events/sessions/pgconfeu2017/session/1559/slides/35/Using%20Kubernetes,%20Docker,%20and%20Helm%20to%20Deploy%20On-Demand%20PostgreSQL%20Streaming%20Replicas.pdf

Postgres Open 2017

- A Kubernetes Operator for PostgreSQL - Architecture and Design
  https://www.sarahconway.com/slides/postgres-operator.pdf
- Containerized Clustered PostgreSQL
  http://jberkus.github.io/container_cluster_pg/#23

PGConf 2017

- Patroni - HA PostgreSQL made easy
  https://www.slideshare.net/AlexanderKukushkin1/patroni-ha-postgresql-made-easy
- PostgreSQL High Availability in a Containerized World
  http://jkshah.blogspot.jp/2017/03/pgconf-2017-postgresql-high.html

1. Introduction

How do people solve this?

PostgreSQL Related Slides and Presentations
https://wiki.postgresql.org/wiki/PostgreSQL_Related_Slides_and_Presentations

PGCONF.EU 2017

- USING KUBERNETES, DOCKER, AND HELM TO DEPLOY ON-DEMAND POSTGRESQL STREAMING REPLICAS
  https://www.postgresql.eu/events/sessions/pgconfeu2017/session/1559/slides/35/Using%20Kubernetes,%20Docker,%20and%20Helm%20to%20Deploy%20On-Demand%20PostgreSQL%20Streaming%20Replicas.pdf

Postgres Open 2017

- A Kubernetes Operator for PostgreSQL - Architecture and Design
  https://www.sarahconway.com/slides/postgres-operator.pdf
- Containerized Clustered PostgreSQL
  http://jberkus.github.io/container_cluster_pg/#23

PGConf 2017

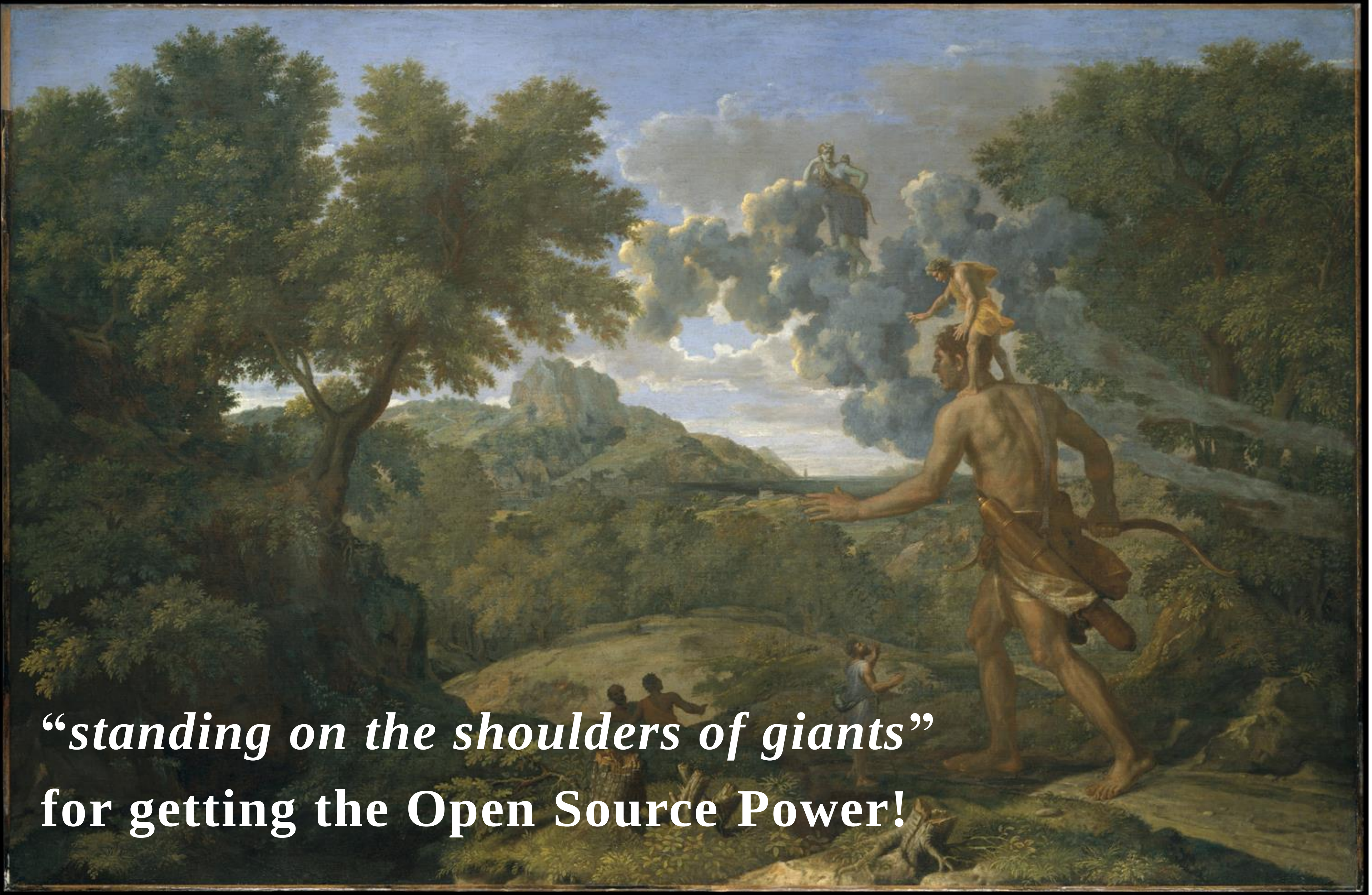
- Patroni - HA PostgreSQL made easy
  https://www.slideshare.net/AlexanderKukushkin1/patroni-ha-postgresql-made-easy
- PostgreSQL High Availability in a Containerized World
  http://jkshah.blogspot.jp/2017/03/pgconf-2017-postgresql-high.html

Container

and

Kubernetes

https://en.wikipedia.org/wiki/Standing_on_the_shoulders_of_giants



“standing on the shoulders of giants”

for getting the Open Source Power!

1. Introduction

PostgreSQL on Kubernetes Projects

- **AtomicDB**
  https://hackernoon.com/postgresql-cluster-into-kubernetes-cluster-f353cde212de
- **Crunchy PostgreSQL Containers (CPC)**
  https://github.com/CrunchyData/crunchy-containers
- **Patroni**
  https://github.com/zalando/patroni
- **Stolon**
  https://github.com/sorintlab/stolon

We are trying CPC on OpenShift origin!

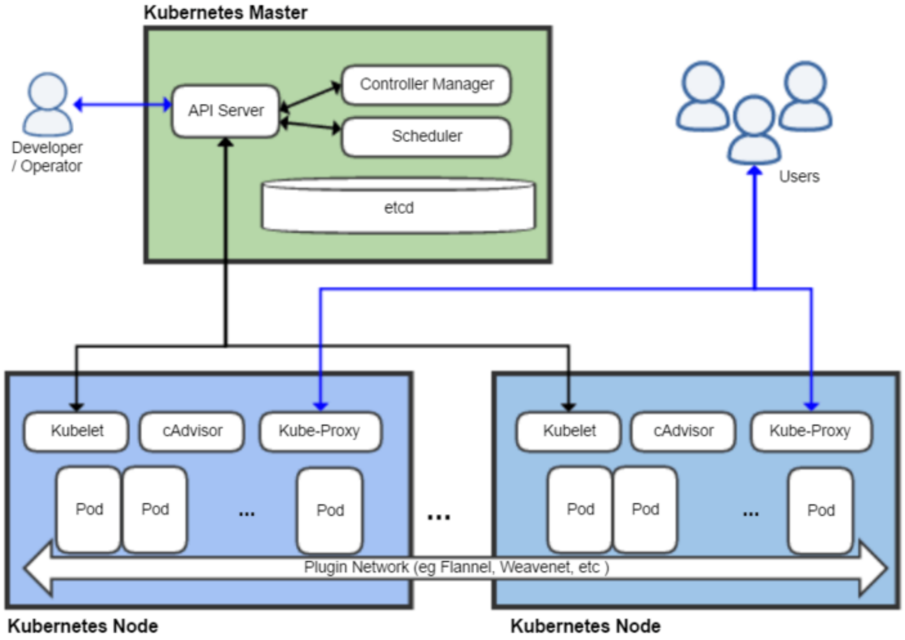
Table of Contents

1. Introduction
2. **Crunchy PostgreSQL Containers on OpenShift**
3. Multi-Master PostgreSQL Cluster on OpenShift
4. Conclusion

1. Introduction

What is Kubernetes?

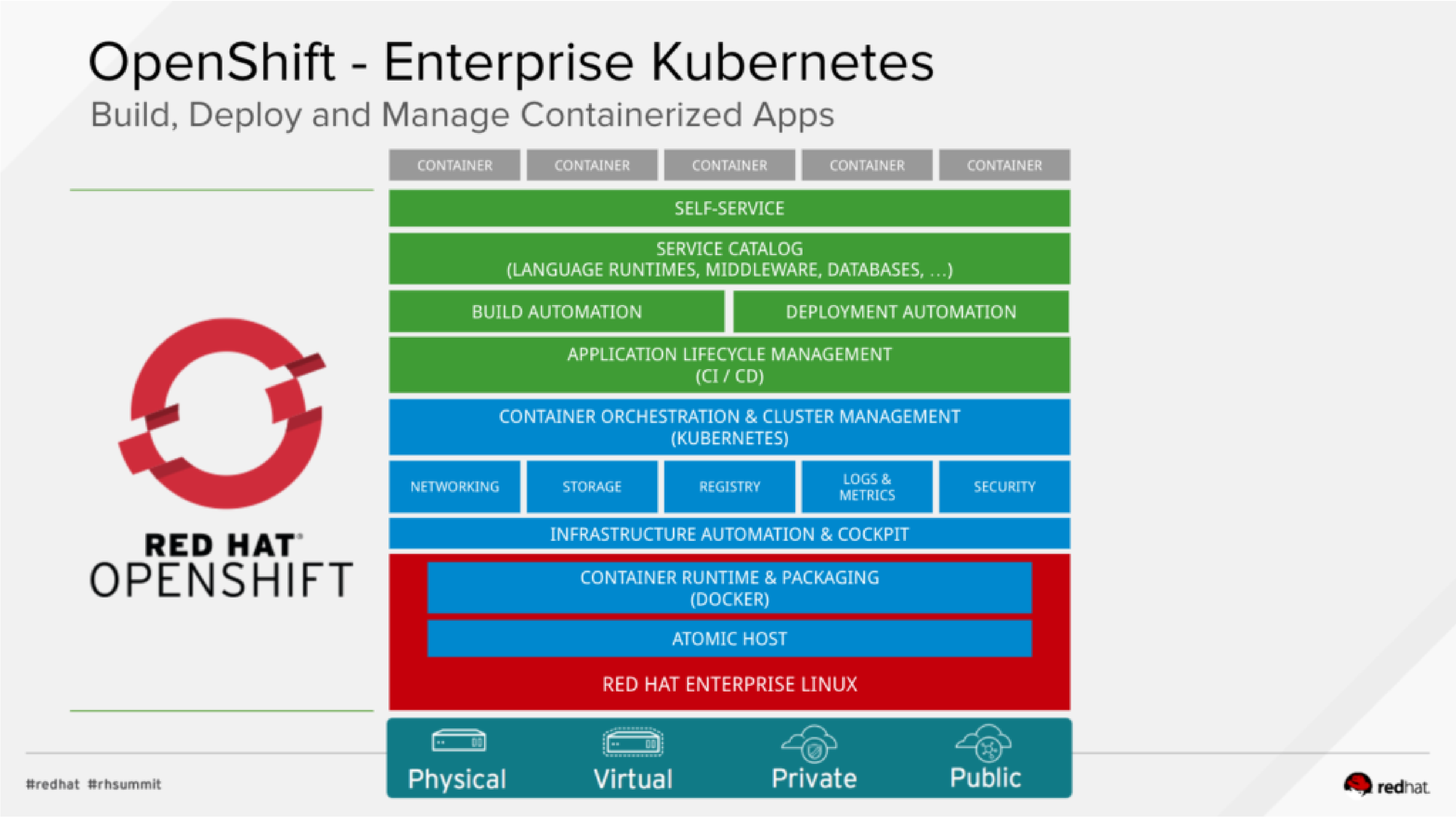
- Kubernetes is an open-source system for management of containerized applications.
- Pod is the basic scheduling unit which consists of one or more containers.
- Kubernetes follows the master-slave architecture.



https://en.wikipedia.org/wiki/Kubernetes

2．Crunchy PostgreSQL Containers on OpenShift

https://blog.openshift.com/enterprise-ready-kubernetes/



OpenShift - Enterprise Kubernetes

Build, Deploy and Manage Containerized Apps

CONTAINER | CONTAINER | CONTAINER | CONTAINER | CONTAINER

SELF-SERVICE

SERVICE CATALOG
(LANGUAGE RUNTIMES, MIDDLEWARE, DATABASES, …)

BUILD AUTOMATION | DEPLOYMENT AUTOMATION

APPLICATION LIFECYCLE MANAGEMENT
(CI / CD)

CONTAINER ORCHESTRATION & CLUSTER MANAGEMENT
(KUBERNETES)

NETWORKING | STORAGE | REGISTRY | LOGS & METRICS | SECURITY

INFRASTRUCTURE AUTOMATION & COCKPIT

CONTAINER RUNTIME & PACKAGING
(DOCKER)

ATOMIC HOST

RED HAT ENTERPRISE LINUX

Physical | Virtual | Private | Public

RED HAT OPENSHIFT

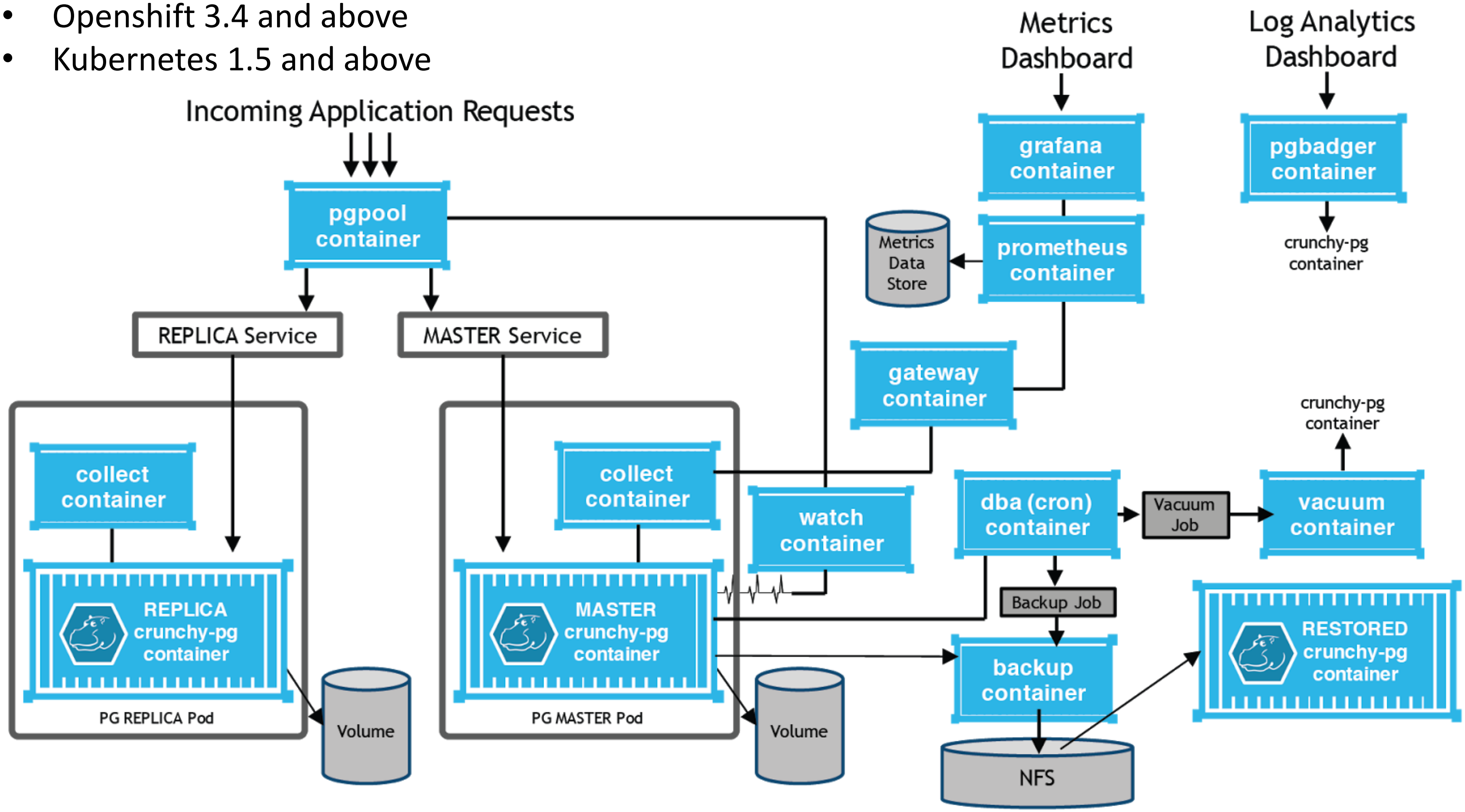
#redhat #rhsummit

redhat

2． Crunchy PostgreSQL Containers on OpenShift

Crunchy PostgreSQL Containers

- Docker 1.12 and above
- Openshift 3.4 and above
- Kubernetes 1.5 and above



https://github.com/CrunchyData/crunchy-containers

2. Crunchy PostgreSQL Containers on OpenShift

How to run a simple PostgreSQL

Write the Pod manifest file.

- simple_postgres_pod.yml

```
kind: Pod
apiVersion: v1
metadata:
  name: simple-pg
  labels:
    name: simple-pg
spec:
  containers:
  - name: postgres
    image: crunchydata/crunchy-postgres:centos7-
10.1-1.7.0
    ports:
    - containerPort: 5432
      protocol: TCP
```

```
   env:
   - name: PGHOST
     value: /tmp
   - name: PG_PRIMARY_USER
     value: primaryuser
   - name: PG_PRIMARY_PORT
     value: '5432'
   - name: PG_MODE
     value: primary
   - name: PG_PRIMARY_PASSWORD
     value: password
   - name: PG_USER
     value: testuser
   - name: PG_PASSWORD
     value: password
   - name: PG_DATABASE
     value: userdb
   - name: PG_ROOT_PASSWORD
     value: password
   volumeMounts:
   - mountPath: /pgdata
     name: pgdata
     readonly: false
  volumes:
  - name: pgdata
   emptyDir: {}
```

Define the Pod state like SQL, not the procedure.

2. Crunchy PostgreSQL Containers on OpenShift

How to run a simple PostgreSQL

Write the Pod manifest file.

- simple_postgres_pod.yml

```
kind: Pod
apiVersion: v1
metadata:
  name: simple-pg
  labels:
    name: simple-pg
spec:
  containers:
  - name: postgres
    image: crunchydata/crunchy-postgres:centos7-10.1-1.7.0
    ports:
    - containerPort: 5432
      protocol: TCP
```

PostgreSQL Container settings

```
    env:
    - name: PGHOST
      value: /tmp
    - name: PG_PRIMARY_USER
      value: primaryuser
    - name: PG_PRIMARY_PORT
      value: '5432'
    - name: PG_MODE
      value: primary
    - name: PG_PRIMARY_PASSWORD
      value: password
    - name: PG_USER
      value: testuser
    - name: PG_PASSWORD
      value: password
    - name: PG_DATABASE
      value: userdb
    - name: PG_ROOT_PASSWORD
      value: password
    volumeMounts:
    - mountPath: /pgdata
      name: pgdata
      readonly: false
  volumes:
  - name: pgdata
    emptyDir: {}
```

Define the Pod state like SQL, not the procedure.

2．Crunchy PostgreSQL Containers on OpenShift

How to run a simple PostgreSQL

Write the Pod manifest file.

- simple_postgres_pod.yml

```
kind: Pod
apiVersion: v1
metadata:
  name: simple-pg
  labels:
    name: simple-pg
spec:
  containers:
  - name: postgres
    image: crunchydata/crunchy-postgres:centos7-
10.1-1.7.0
    ports:
    - containerPort: 5432
      protocol: TCP
```

PostgreSQL Container settings

pull the container image from DockerHub

```
  env:
  - name: PGHOST
    value: /tmp
  - name: PG_PRIMARY_USER
    value: primaryuser
  - name: PG_PRIMARY_PORT
    value: '5432'
  - name: PG_MODE
    value: primary
  - name: PG_PRIMARY_PASSWORD
    value: password
  - name: PG_USER
    value: testuser
  - name: PG_PASSWORD
    value: password
  - name: PG_DATABASE
    value: userdb
  - name: PG_ROOT_PASSWORD
    value: password
  volumeMounts:
  - mountPath: /pgdata
    name: pgdata
    readonly: false
volumes:
- name: pgdata
  emptyDir: {}
```

Define the Pod state like SQL, not the procedure.

2. Crunchy PostgreSQL Containers on OpenShift

How to run a simple PostgreSQL

Write the Pod manifest file.

- simple_postgres_pod.yml

```
kind: Pod
apiVersion: v1
metadata:
  name: simple-pg
  labels:
    name: simple-pg
spec:
  containers:
  - name: postgres
    image: crunchydata/crunchy-postgres:centos7-
10.1-1.7.0
    ports:
    - containerPort: 5432
      protocol: TCP
```

PostgreSQL Container settings

pull the container image from DockerHub

PostgreSQL settings

```
    env:
    - name: PGHOST
      value: /tmp
    - name: PG_PRIMARY_USER
      value: primaryuser
    - name: PG_PRIMARY_PORT
      value: '5432'
    - name: PG_MODE
      value: primary
    - name: PG_PRIMARY_PASSWORD
      value: password
    - name: PG_USER
      value: testuser
    - name: PG_PASSWORD
      value: password
    - name: PG_DATABASE
      value: userdb
    - name: PG_ROOT_PASSWORD
      value: password
    volumeMounts:
    - mountPath: /pgdata
      name: pgdata
      readonly: false
  volumes:
  - name: pgdata
    emptyDir: {}
```

Define the Pod state like SQL, not the procedure.

2. Crunchy PostgreSQL Containers on OpenShift

How to run a simple PostgreSQL

Write the Pod manifest file.

- simple_postgres_pod.yml

```
kind: Pod
apiVersion: v1
metadata:
  name: simple-pg
  labels:
    name: simple-pg
spec:
  containers:
  - name: postgres
    image: crunchydata/crunchy-postgres:centos7-
10.1-1.7.0
    ports:
    - containerPort: 5432
      protocol: TCP
```

PostgreSQL Container settings

pull the container image from DockerHub

```
    env:
    - name: PGHOST
      value: /tmp
    - name: PG_PRIMARY_USER
      value: primaryuser
    - name: PG_PRIMARY_PORT
      value: '5432'
    - name: PG_MODE
      value: primary
    - name: PG_PRIMARY_PASSWORD
      value: password
    - name: PG_USER
      value: testuser
    - name: PG_PASSWORD
      value: password
    - name: PG_DATABASE
      value: userdb
    - name: PG_ROOT_PASSWORD
      value: password
    volumeMounts:
    - mountPath: /pgdata
      name: pgdata
      readonly: false
  volumes:
  - name: pgdata
    emptyDir: {}
```

PostgreSQL settings

Pod storage settings

Define the Pod state like SQL, not the procedure.

2．Crunchy PostgreSQL Containers on OpenShift

How to run a simple PostgreSQL

Write the Service manifest file.

- simple_postgres_svc.yml

```
kind: Service
apiVersion: v1
metadata:
  name: simple-pg
  labels:
    name: simple-pg
spec:
  ports:
  - protocol: TCP
    port: 5432
    targetPort: 5432
    nodePort: 0
  selector:
    name: simple-pg
  type: ClusterIP
  sessionAffinity: None
status:
  loadBalancer: {}
```

2．Crunchy PostgreSQL Containers on OpenShift

How to run a simple PostgreSQL

Write the Service manifest file.

- simple_postgres_svc.yml

```
kind: Service
apiVersion: v1
metadata:
  name: simple-pg
  labels:
    name: simple-pg
spec:
  ports:
  - protocol: TCP
    port: 5432
    targetPort: 5432
    nodePort: 0
  selector:
    name: simple-pg
  type: ClusterIP
  sessionAffinity: None
status:
  loadBalancer: {}
```

This Service sends packets to TCP port 5432 on any Pod with the "name: simple-pg" label.

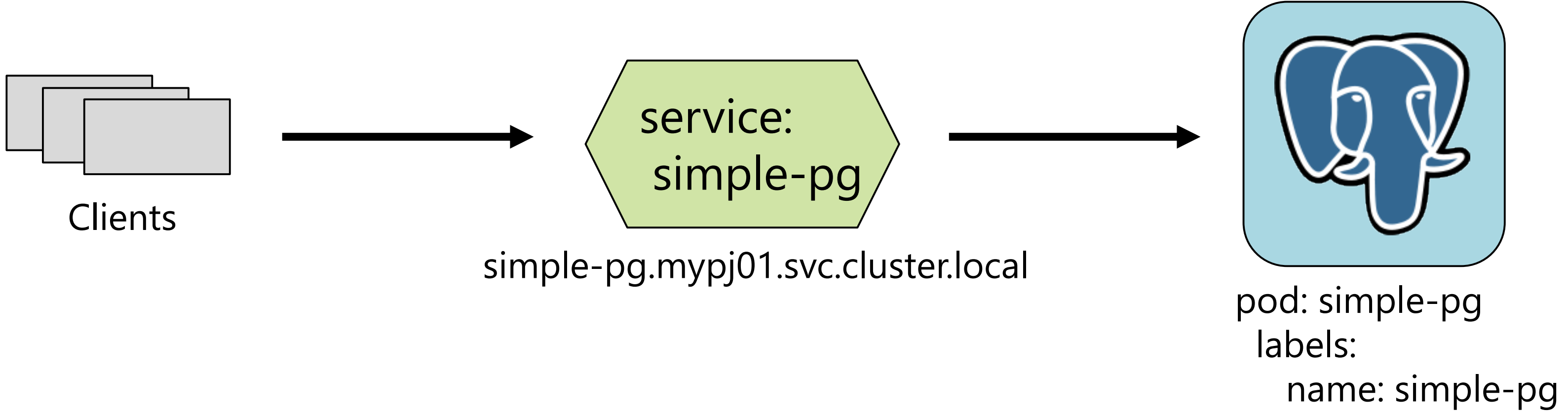
2. Crunchy PostgreSQL Containers on OpenShift

How to run a simple PostgreSQL

Execute “**oc create**” to create the **Simple PostgreSQL pod and service**.

```
$ oc create -f simple_postgres_svc.yml
service "simple-pg" created

$ oc create -f simple_postgres_pod.yml
pod "simple-pg" created
```

The simple-pg service sends SQLs to the simple-pg pod.



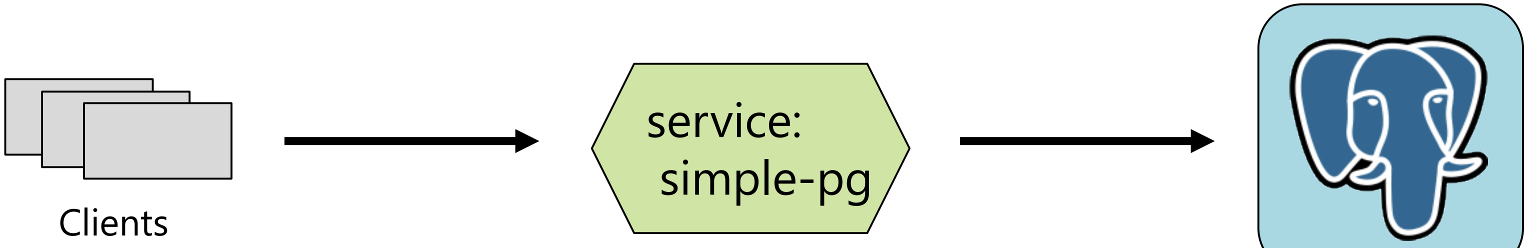
pod: simple-pg
  labels:
    name: simple-pg

2．Crunchy PostgreSQL Containers on OpenShift

How to run a simple PostgreSQL

Login to the hostname “**simple-pg.mypj01.svc.cluster.local**”.

```
$ psql -h simple-pg.mypj01.svc.cluster.local -U testuser -d userdb
Password for user testuser:

userdb=> ¥d
                List of relations
  Schema  |        Name         | Type  |  Owner
----------+---------------------+-------+----------
 public   | pg_stat_statements | view  | postgres
 testuser | testtable           | table | testuser
```

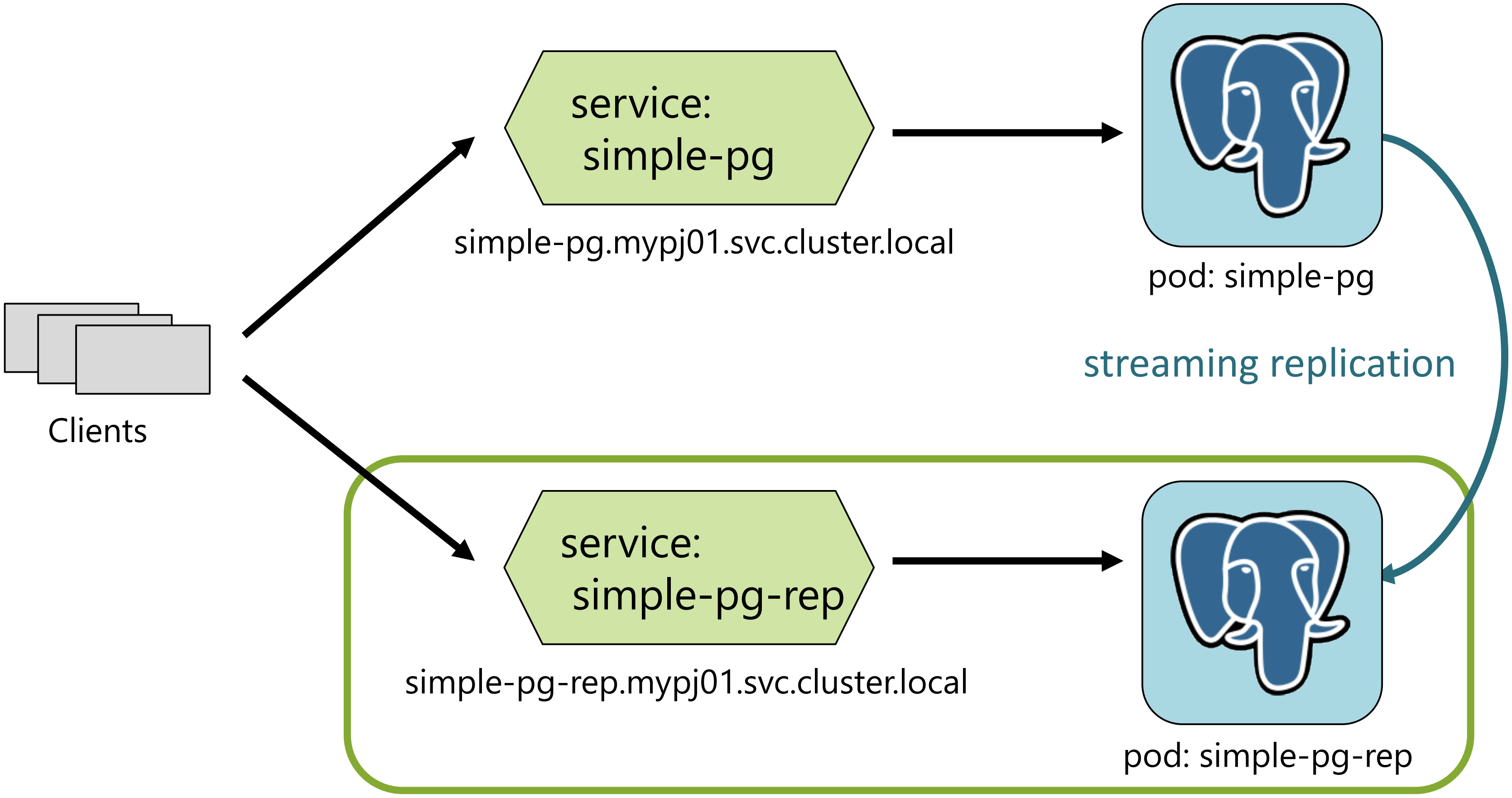


simple-pg.mypj01.svc.cluster.local

pod: simple-pg
labels:
name: simple-pg

2．Crunchy PostgreSQL Containers on OpenShift

Let’s try to add a replication!



Add replication Pod and Service

2. Crunchy PostgreSQL Containers on OpenShift

Let's try to add a replication!

Write the Pod and Service manifest file.

- simple_postgres_pod-rep.yml

```
kind: Pod
apiVersion: v1
metadata:
  name: simple-pg-rep
  labels:
    name: simple-pg-rep
spec:
  containers:
  - name: postgres
    image: crunchydata/crunchy-postgres:centos7-
10.1-1.7.0
    ports:
    - containerPort: 5432
      protocol: TCP
```

Crunchy-postgres runs a shell script for initialization when the container starts.
The PG_MODE variable is used to determine the role(Primary Postgres or Replica Postgres).

```
    env:
    - name: PGHOST
      value: /tmp
    - name: PG_PRIMARY_USER
      value: primaryuser
    - name: PG_PRIMARY_PORT
      value: '5432'
    - name: PG_MODE
      value: replica
    - name: PG_PRIMARY_PASSWORD
      value: password
    - name: PG_USER
      value: testuser
    - name: PG_PASSWORD
      value: password
    - name: PG_DATABASE
      value: userdb
    - name: PG_ROOT_PASSWORD
      value: password
    - name: PG_PRIMARY_HOST
      value: simple-pg
    - name: PG_PRIMARY_PORT
      value: '5432'
    volumeMounts:
    - mountPath: /pgdata
      name: pgdata
      readOnly: false
  volumes:
  - name: pgdata
    emptyDir: {}
```

2．Crunchy PostgreSQL Containers on OpenShift

Let’s try to add a replication!

Execute “**oc create**” to create the **Replication pod and service**.

```
$ oc create -f simple_postgres_rep_svc.yml
service "simple-pg-rep" created

$ oc create -f simple_postgres_rep_pod.yml
pod "simple-pg-rep" created
```

2． Crunchy PostgreSQL Containers on OpenShift

Let’s try to add a replication!

Check replication status

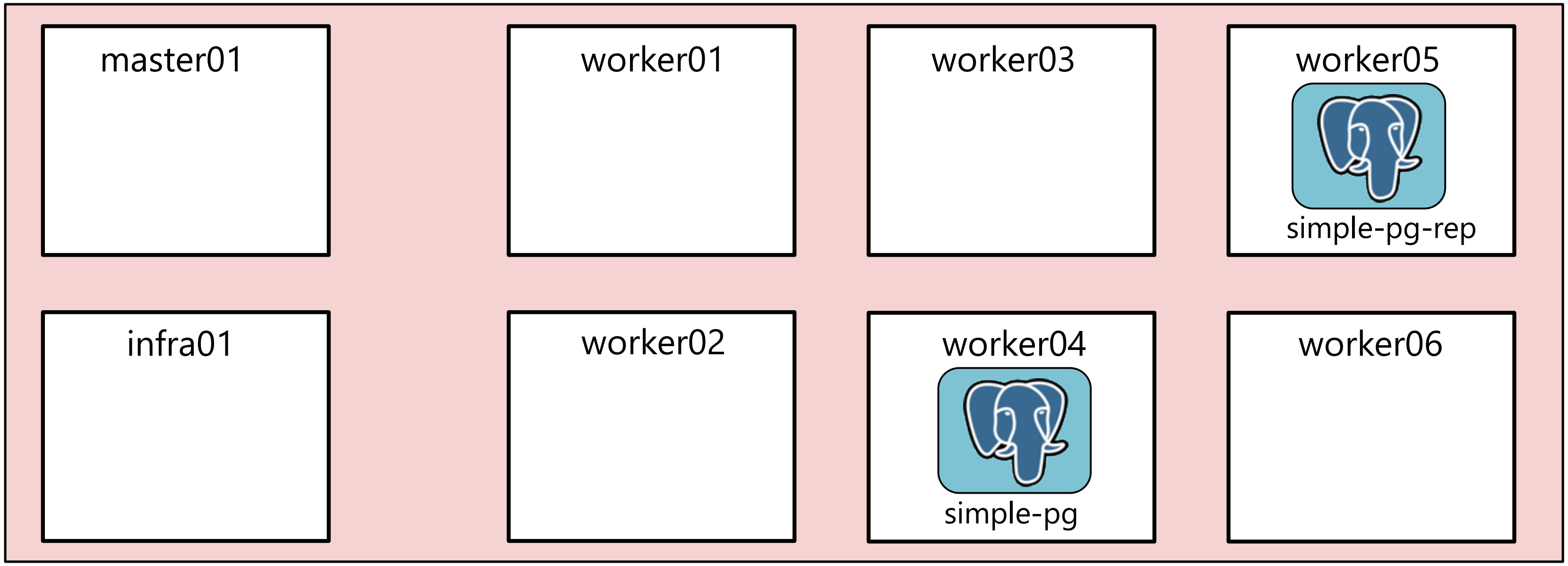
```
$ psql -h simple-pg.mypj01.svc.cluster.local -U postgres -d userdb
userdb=# select * from pg_stat_replication;
-[ RECORD 1 ]----+------------------------------
pid              | 329
usesysid         | 16391
usename          | primaryuser
application_name | simple-pg-rep
client_addr      | 10.131.0.1
...
sync_state       | async
```

2. Crunchy PostgreSQL Containers on OpenShift

Where is PostgreSQL running on OpenShif Cluster?

```
$ oc get pod -o wide
NAME                READY   STATUS    RESTARTS  AGE    IP            NODE
simple-pg           1/1     Running   0         7m     10.129.2.20   worker04
simple-pg-rep       1/1     Running   0         5m     10.131.0.16   worker05
```

OpenShift Cluster (8 virtual machines)



We don’t have to manage these IP addresses and where these containers are located!

2．Crunchy PostgreSQL Containers on OpenShift

At first, what should we think about when running PostgreSQL on OpenShift(Kubernetes)?

- **What should we use for a Persistent storage?**
    - Local disk
    - (Shared) Network storage
- **What should we do in case of server/container failures?**
    - Failover
    - Restart PostgreSQL

2. Crunchy PostgreSQL Containers on OpenShift

At first, what should we think about when running PostgreSQL on OpenShift(Kubernetes)?

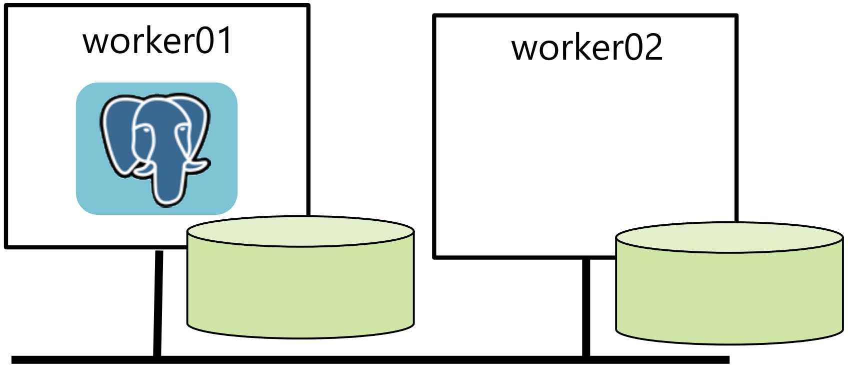
- **What should we use for a Persistent storage?**
  - Local disk
  - (Shared) Network storage
- **What should we do in case of server/container failures?**
  - Failover
  - Restart PostgreSQL

2. Crunchy PostgreSQL Containers on OpenShift

What should we use for storage?

Local disk

Each worker node has several HDD/SSD.

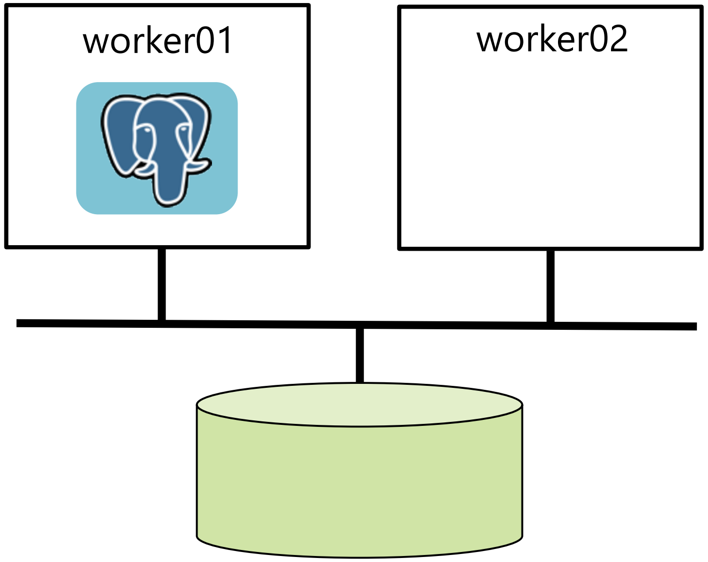


2. Crunchy PostgreSQL Containers on OpenShift

What should we use for storage?

(Shared) Network storage

Each worker node connects to one or more network storages.



PersistentVolume plug-in

OpenShift Origin supports the following PersistentVolume plug-ins:

- NFS
- HostPath
- GlusterFS
- Ceph RBD
- OpenStack Cinder
- AWS Elastic Block Store (EBS)
- GCE Persistent Disk
- iSCSI
- Fibre Channel
- Azure Disk
- Azure File
- VMWare vSphere
- Local

NFS example

```
apiVersion: v1
 kind: PersistentVolume
 metadata:
   name: pv0003
 spec:
   capacity:
     storage: 5Gi
   accessModes:
     - ReadWriteOnce
   persistentVolumeReclaimPolicy: Recycle
   nfs:
     path: /tmp
     server: 172.17.0.2
```

https://docs.openshift.org/latest/architecture/additional_concepts/storage.html

2． Crunchy PostgreSQL Containers on OpenShift

At first, what should we think about when running PostgreSQL on OpenShift(Kubernetes)?

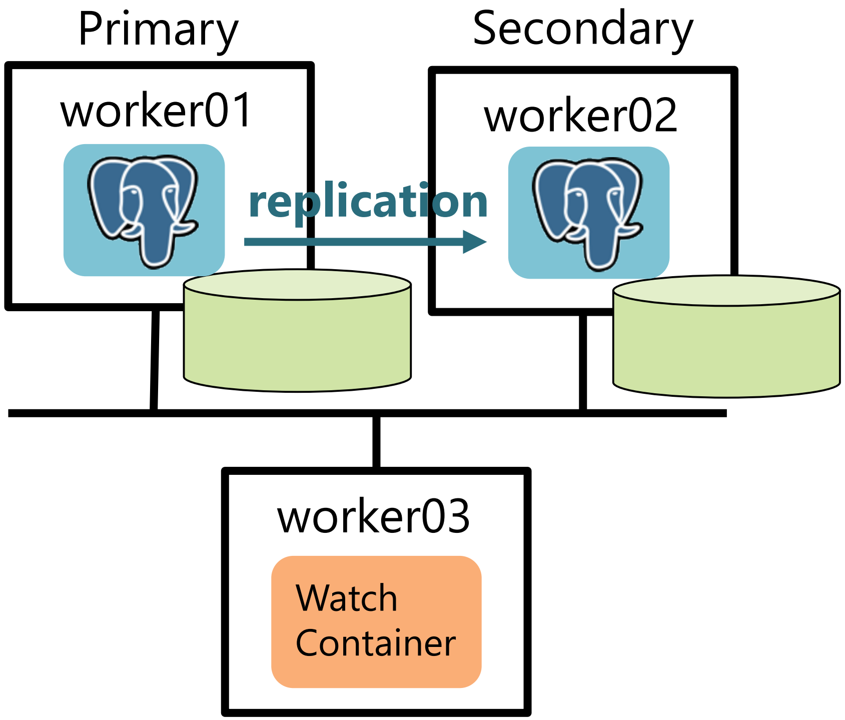
- **What should we use for a Persistent storage?**
  - Local disk
  - (Shared) Network storage

- **What should we do in case of server/container failures?**
  - Failover
  - Restart PostgreSQL

2．Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Failover**
  CPC provides a Watch Container to manage failover.

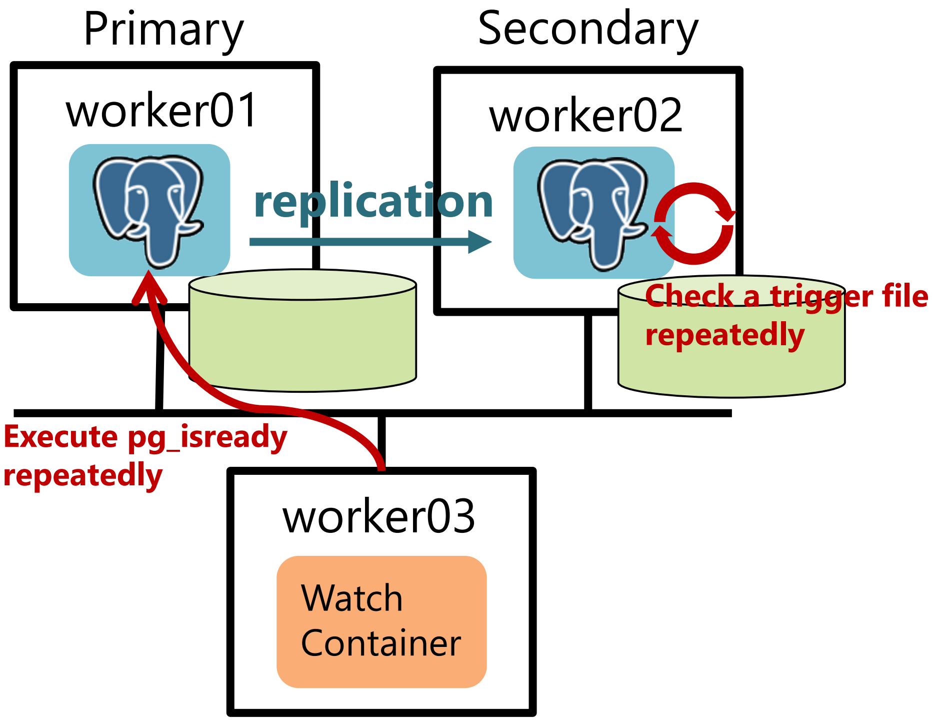


2. Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Failover**

  CPC provides a Watch Container to manage failover.

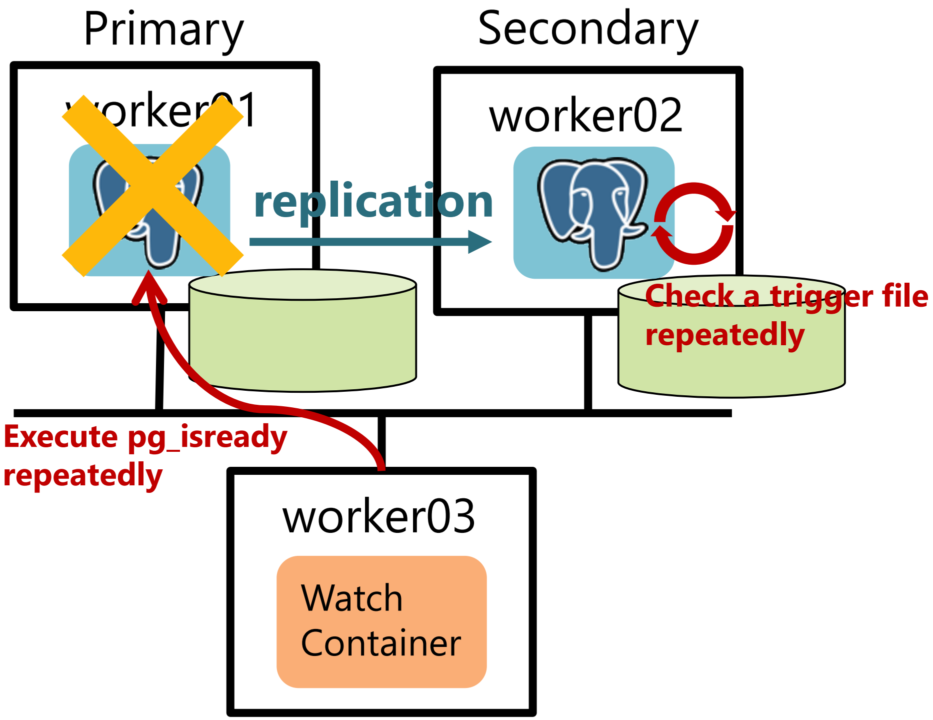


2． Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Failover**

  CPC provides a Watch Container to manage failover.

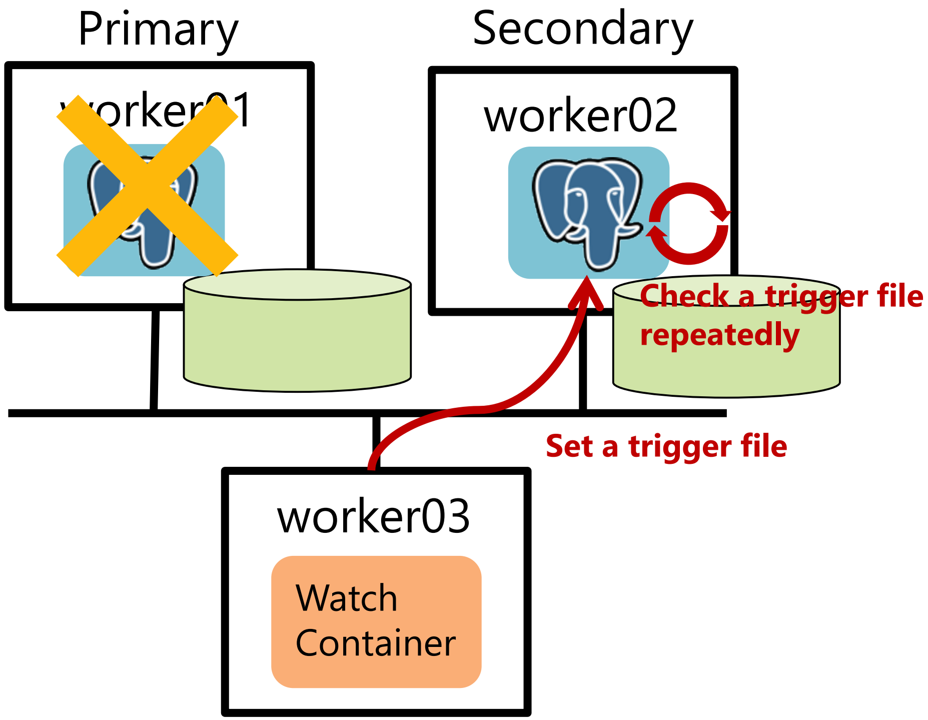


2． Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Failover**

  CPC provides a Watch Container to manage failover.

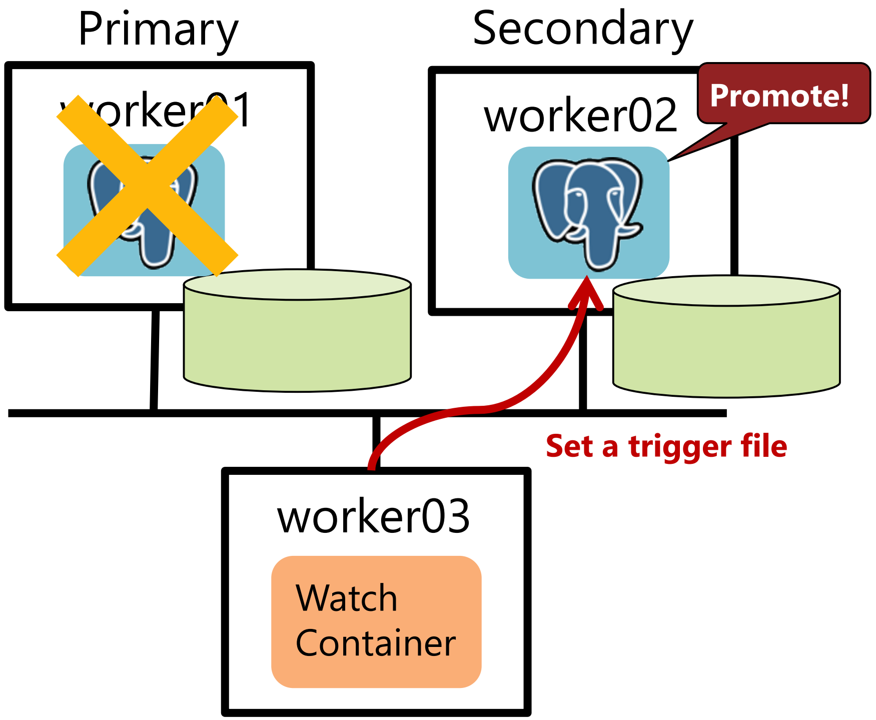


2. Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Failover**

  CPC provides a Watch Container to manage failover.



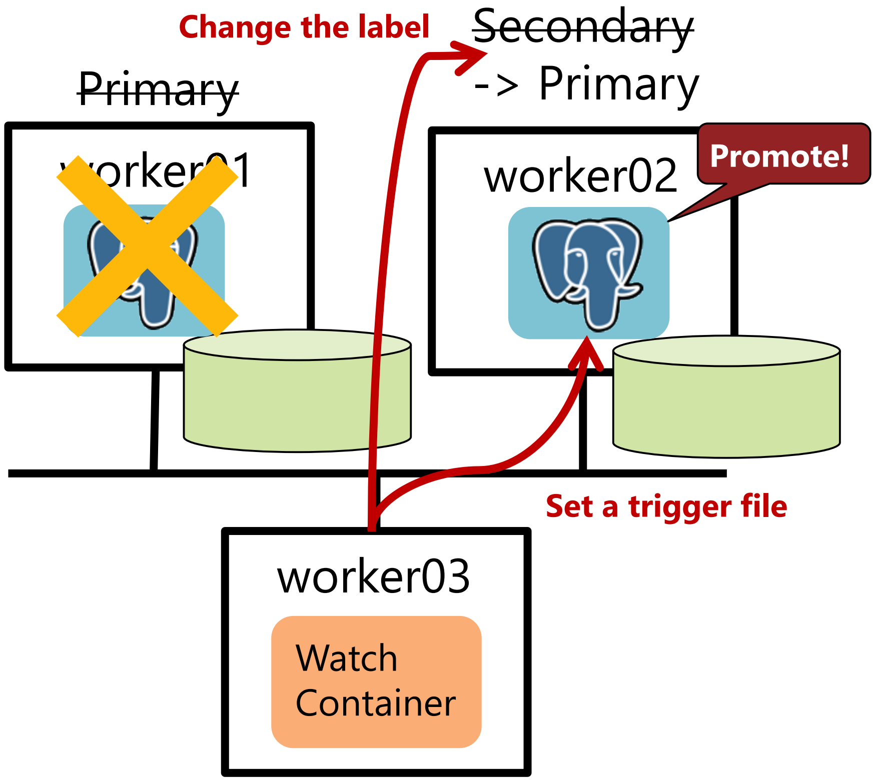
Copyright©2017 NTT corp. All Rights Reserved.

2. Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Failover**

  CPC provides a Watch Container to manage failover.

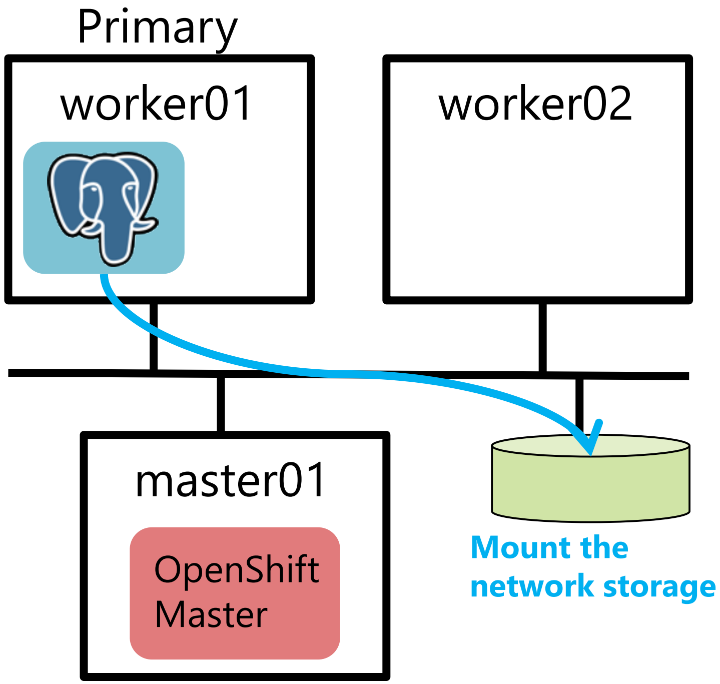


2. Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Restart PostgreSQL**

  OpenShift(Kubernetes) provides Liveness Probe to confirm whether the container is alive.

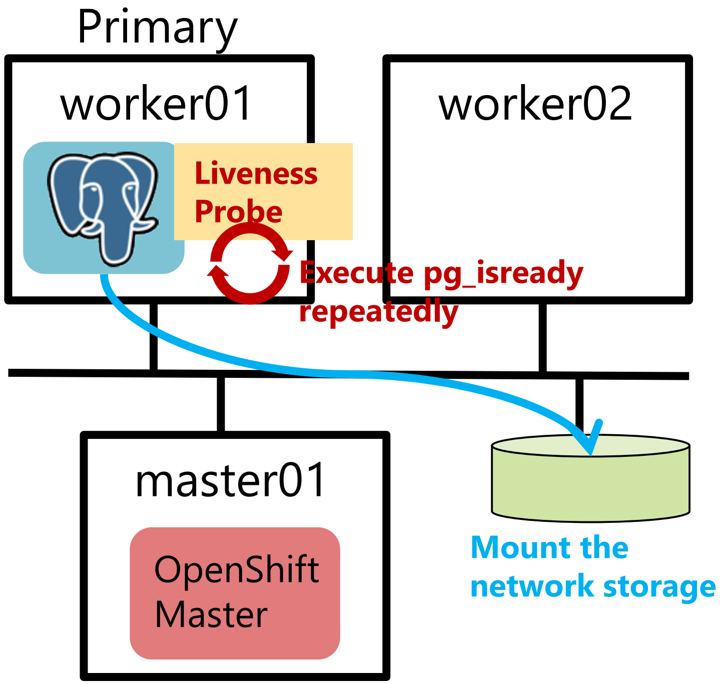


Copyright©2017 NTT corp. All Rights Reserved.

2. Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

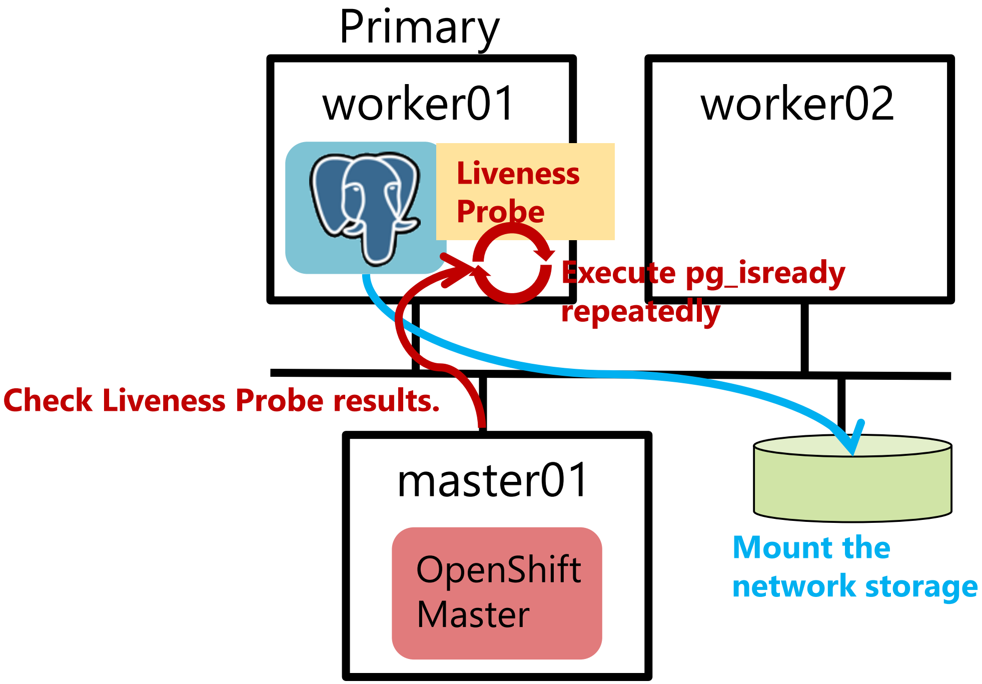
- **Restart PostgreSQL**
  OpenShift(Kubernetes) provides Liveness Probe to confirm whether the container is alive.



2． Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Restart PostgreSQL**
  OpenShift(Kubernetes) provides Liveness Probe to confirm whether the container is alive.

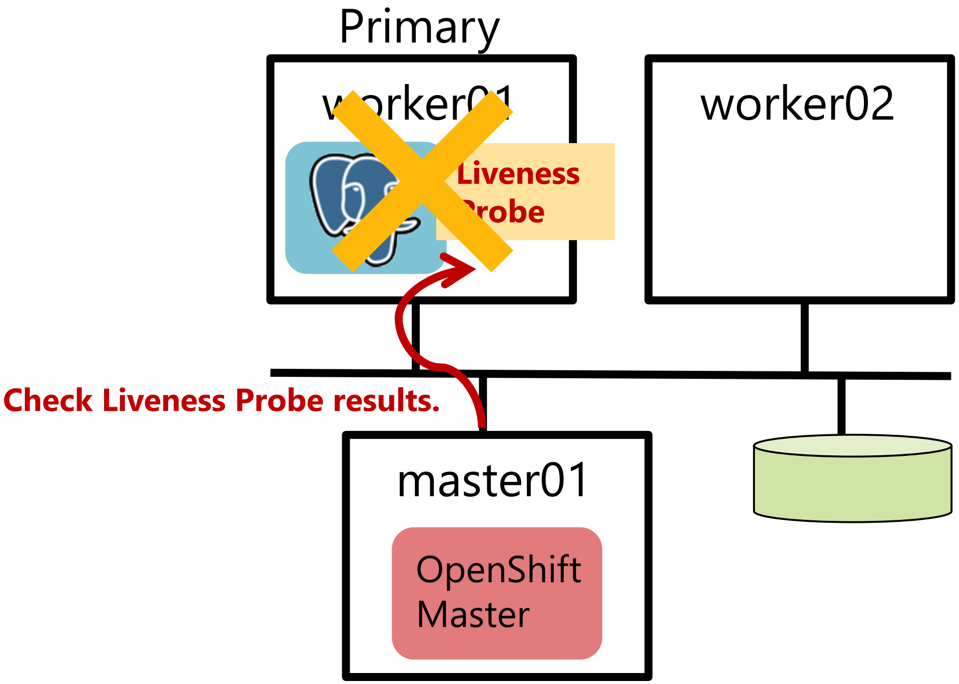

Primary
worker01
Liveness Probe
worker02
Execute pg_isready repeatedly
Check Liveness Probe results.
master01
OpenShift Master
Mount the network storage

2. Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Restart PostgreSQL**

  OpenShift(Kubernetes) provides Liveness Probe to confirm whether the container is alive.

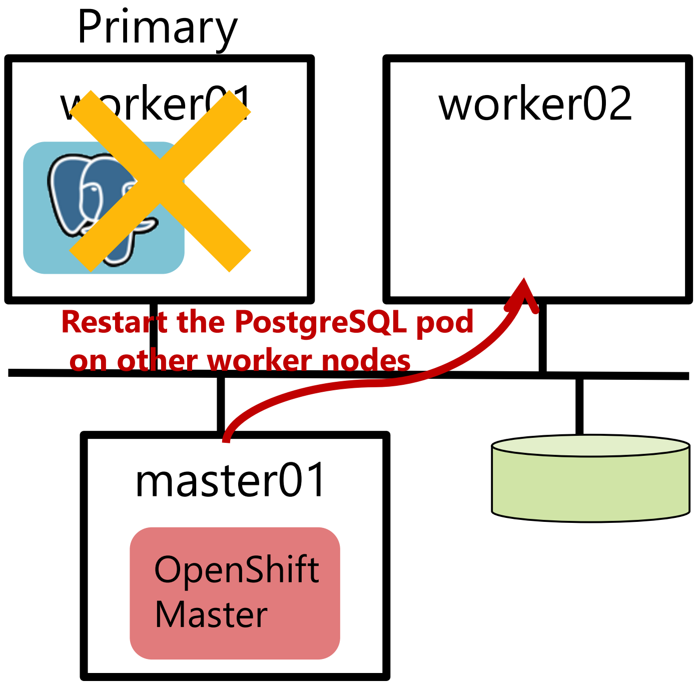


2. Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Restart PostgreSQL**

  OpenShift(Kubernetes) provides Liveness Probe to confirm whether the container is alive.

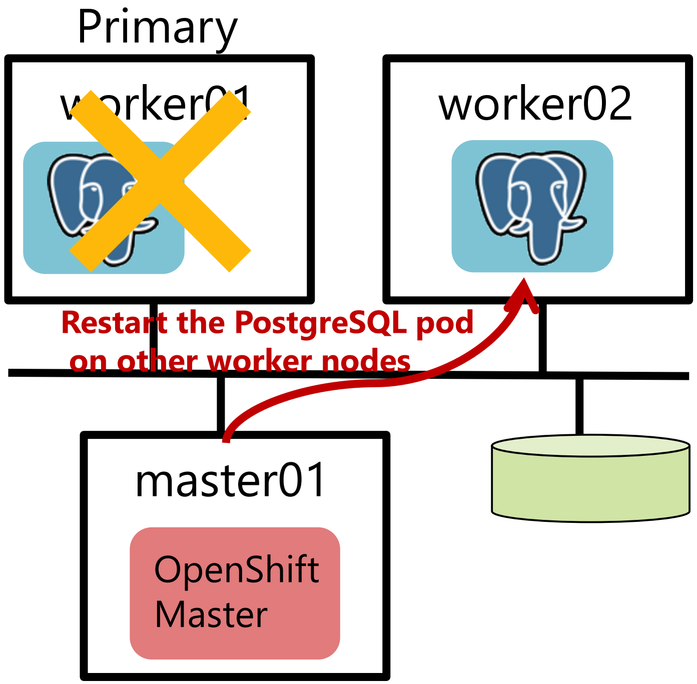


2． Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Restart PostgreSQL**

  OpenShift(Kubernetes) provides Liveness Probe to confirm whether the container is alive.

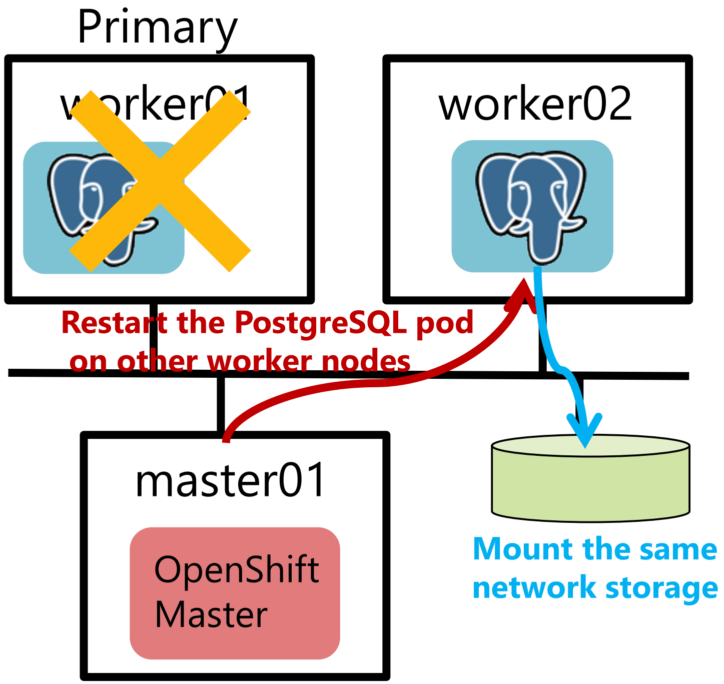


2. Crunchy PostgreSQL Containers on OpenShift

What should we do in case of server/container failures?

- **Restart PostgreSQL**

  OpenShift(Kubernetes) provides Liveness Probe to confirm whether the container is alive.



2．Crunchy PostgreSQL Containers on OpenShift

Choose a suitable method for your environments.

Persistent storage

- Local disk
- (Shared) Network storage

in case of server/container failures

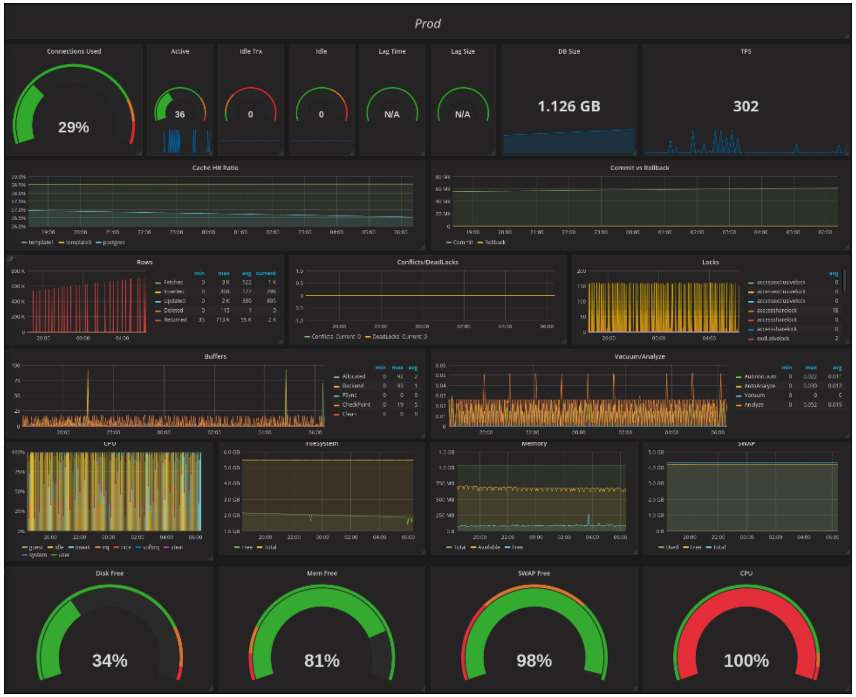
- Failover
- Restart PostgreSQL

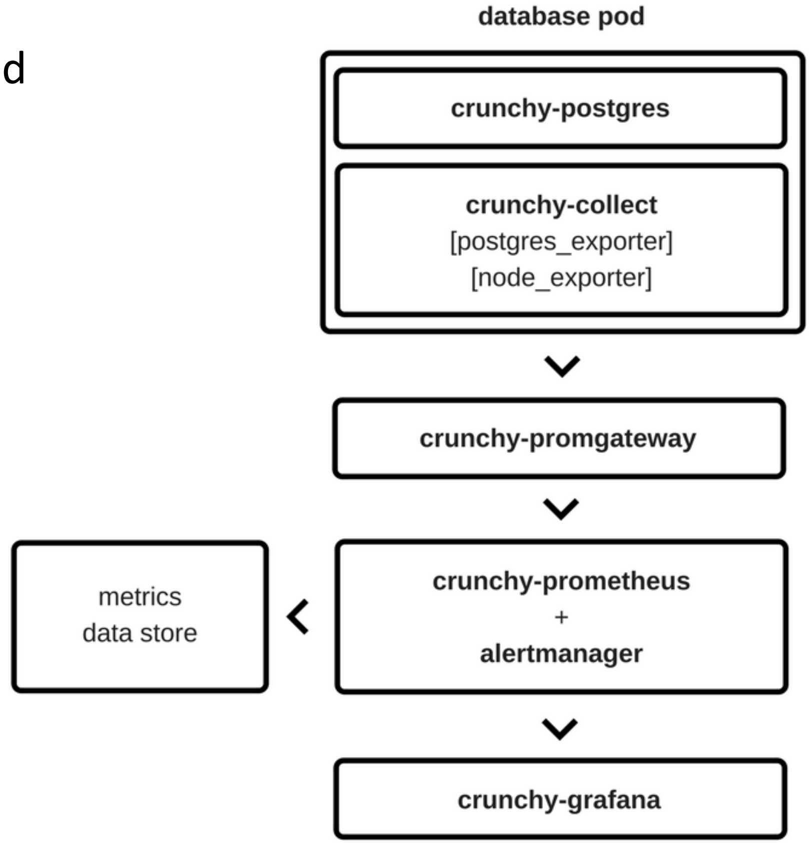
CPC other examples

crunchy-collect

https://github.com/CrunchyData/crunchy-containers/blob/master/docs/metrics.adoc

The crunchy-collect container gathers different metrics from the crunchy-postgres container and pushes these to the Prometheus Promgateway.




database pod

crunchy-postgres

crunchy-collect [postgres_exporter] [node_exporter]

crunchy-promgateway

crunchy-prometheus + alertmanager

metrics data store

crunchy-grafana

2． Crunchy PostgreSQL Containers on OpenShift

Advantages of PostgreSQL on Kubernetes

Easy to run PostgreSQL (including HA, monitoring etc.).

- ➢Define the state, not the procedure. The procedure is packed into the container with portability.
- ➢The network is abstracted. This makes easier to define the state.

Disadvantages of PostgreSQL on Kubernetes

Hard to manage Kubernetes cluster. (maybe...)

- ➢It seems to me that the complexity of PostgreSQL layer is shifted to Kubernetes layer.
- ➢But if you are already running a Kubernetes cluster (and that will come soon?) I believe this will not be a problem.

Table of Contents

1. Introduction
2. Crunchy PostgreSQL Containers on OpenShift
3. **Multi-Master PostgreSQL Cluster on OpenShift**
4. Conclusion

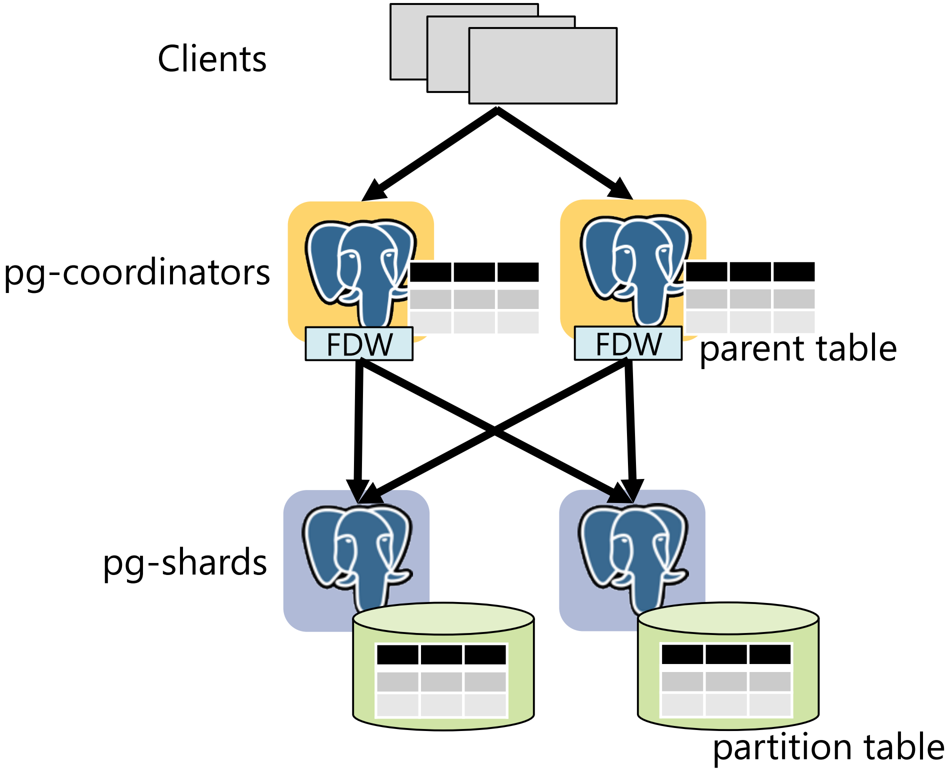
3. Multi-Master PostgreSQL Cluster on OpenShift

Try to install Built-in Sharding base Multi-Master PostgreSQL Cluster on OpenShift.

- The PostgreSQL Container contains the same source code which was demonstrated in the today's morning session "Built-in Sharding Special Version” and the modified scripts of Crunchy-postgres container.

  $ docker pull ooyamams/postgres-dev

- Using NFS as a shared network storage for test purposes.
- pg-coordinator Pod and pg-shard Pod are created by Templates and controlled by DeploymentConfig, not StatefulSet.



3． Multi-Master PostgreSQL

pg-coordinator.yml

```
kind: Template
apiVersion: v1
metadata:
  name: pg-coordinator
  creationTimestamp: null
  annotations:
    description: PostgreSQL Multi-Master Build in Sharding Example
    iconClass: icon-database
    tags: database,postgresql
parameters:
- name: PG_PRIMARY_USER
  description: The username used for primary / replica replication
  value: primaryuser
  ...(omit)...
objects:
- kind: Service
  apiVersion: v1
 ...(omit)...
- kind: DeploymentConfig
  apiVersion: v1
  metadata:
    name: ${PG_PRIMARY_SERVICE_NAME}
    creationTimestamp: null
  spec:
    strategy:
      type: Recreate
      resources: {}
    triggers:
    - type: ConfigChange
    replicas: 2
```

```
    selector:
      name: ${PG_PRIMARY_SERVICE_NAME}
    template:
      metadata:
        creationTimestamp: null
        labels:
          name: ${PG_PRIMARY_SERVICE_NAME}
      spec:
        serviceAccount: pg-cluster-sa
        containers:
        - name: server
          image: 172.30.81.49:5000/mypj01/postgres-dev:build-in-sharding-4
          livenessProbe:
            exec:
              command:
              - /opt/cpm/bin/liveness.sh
            initialDelaySeconds: 90
            timeoutSeconds: 1
          ports:
          - containerPort: 5432
            protocol: TCP
          env:
          - name: PG_PRIMARY_HOST
            value: ${PG_PRIMARY_SERVICE_NAME}
 ...(omit)...
          resources: {}
          terminationMessagePath: /dev/termination-log
          securityContext:
            privileged: false
          volumeMounts:
          - mountPath: /pgdata
            name: pgdata
            readOnly: false
        volumes:
        - name: pgdata
          emptyDir: {}
        restartPolicy: Always
        dnsPolicy: ClusterFirst
  status: {}
```

3． Multi-Master PostgreSQL Cluster on OpenShift

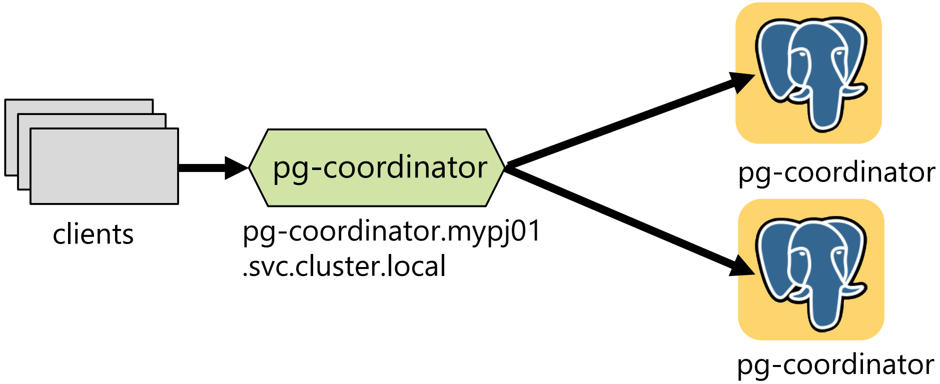
Try to install Built-in Sharding base Multi-Master PostgreSQL Cluster on OpenShift.

- Create Template for pg-coordinator.

```
$ oc create -f pg-coordinator.yml
```

- Create Service and DeploymentConfig from the Template.

```
$ oc process pg-coordinator | oc create -f -
```



3． Multi-Master PostgreSQL Cluster on OpenShift

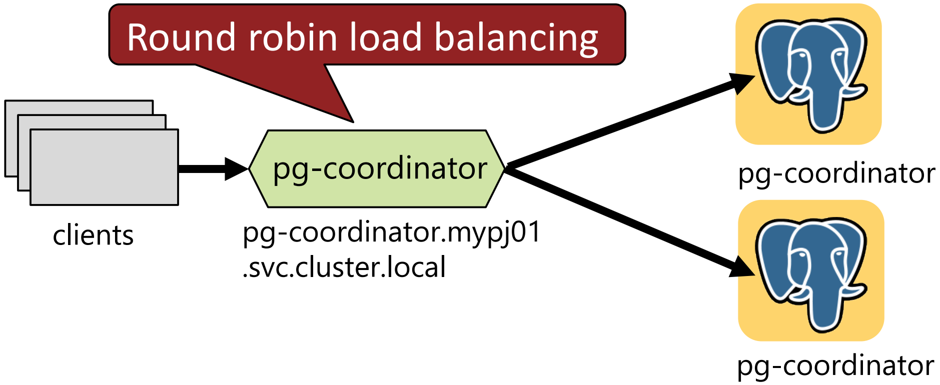
Try to install Built-in Sharding base Multi-Master PostgreSQL Cluster on OpenShift.

- Create Template for pg-coordinator.

```
$ oc create -f pg-coordinator.yml
```

- Create Service and DeploymentConfig from the Template.

```
$ oc process pg-coordinator | oc create -f -
```



3. Multi-Master PostgreSQL

pg-shard.yml

```
kind: Template
apiVersion: v1
metadata:
  name: pg-shared
  creationTimestamp: null
  annotations:
    description: PostgreSQL Multi-Master Build in Sharding Example
    iconClass: icon-database
    tags: database,postgresql
parameters:
- name: PG_PRIMARY_USER
  description: The username used for primary / replica replication
  value: primaryuser
 ...(omit)...
- name: PGDATA_PATH_OVERRIDE
  value: shared-01
objects:
- kind: Service
  apiVersion: v1
  metadata:
    name: ${PG_PRIMARY_SERVICE_NAME}
    labels:
      name: ${PG_PRIMARY_SERVICE_NAME}
  spec:
    ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
      nodePort: 0
    selector:
      name: ${PG_PRIMARY_SERVICE_NAME}
    clusterIP: None
```

```
- kind: DeploymentConfig
  apiVersion: v1
  metadata:
    name: ${PG_PRIMARY_SERVICE_NAME}
    creationTimestamp: null
  spec:
    serviceName: ${PG_PRIMARY_SERVICE_NAME}
    strategy:
 ...(omit)...
        labels:
          name: ${PG_PRIMARY_SERVICE_NAME}
      spec:
        serviceAccount: pg-sa
        containers:
        - name: ${PG_PRIMARY_SERVICE_NAME}
          image: 172.30.81.49:5000/mypj01/postgres-dev:build-in-sharding-1
          livenessProbe:
            exec:
              command:
              - /opt/cpm/bin/liveness.sh
            initialDelaySeconds: 90
            timeoutSeconds: 1
          ports:
          - containerPort: 5432
            protocol: TCP
          env:
          - name: PG_PRIMARY_HOST
            value: ${PG_PRIMARY_SERVICE_NAME}
 ...(omit)...
          resources: {}
          terminationMessagePath: /dev/termination-log
          securityContext:
            privileged: false
          volumeMounts:
          - name: pgdata
            mountPath: /pgdata
            readOnly: false
        volumes:
        - name: pgdata
          persistentVolumeClaim:
            claimName: crunchy-pvc
```

3. Multi-Master PostgreSQL Cluster on OpenShift

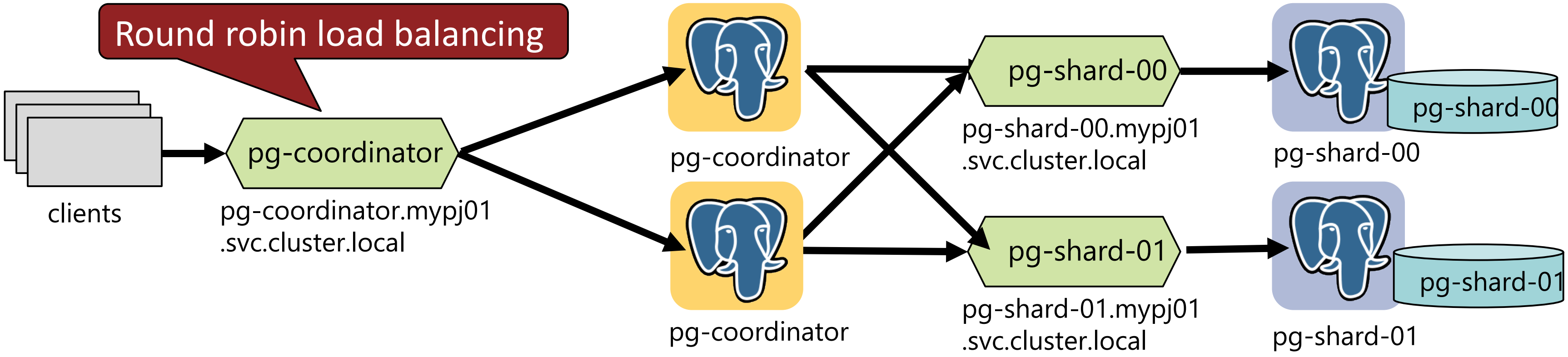
Try to install Built-in Sharding base Multi-Master PostgreSQL Cluster on OpenShift.

- Create Template for pg-shard.

```
$ oc create -f pg-shard.yml
```

- Create Service and DeploymentConfig from the Template.

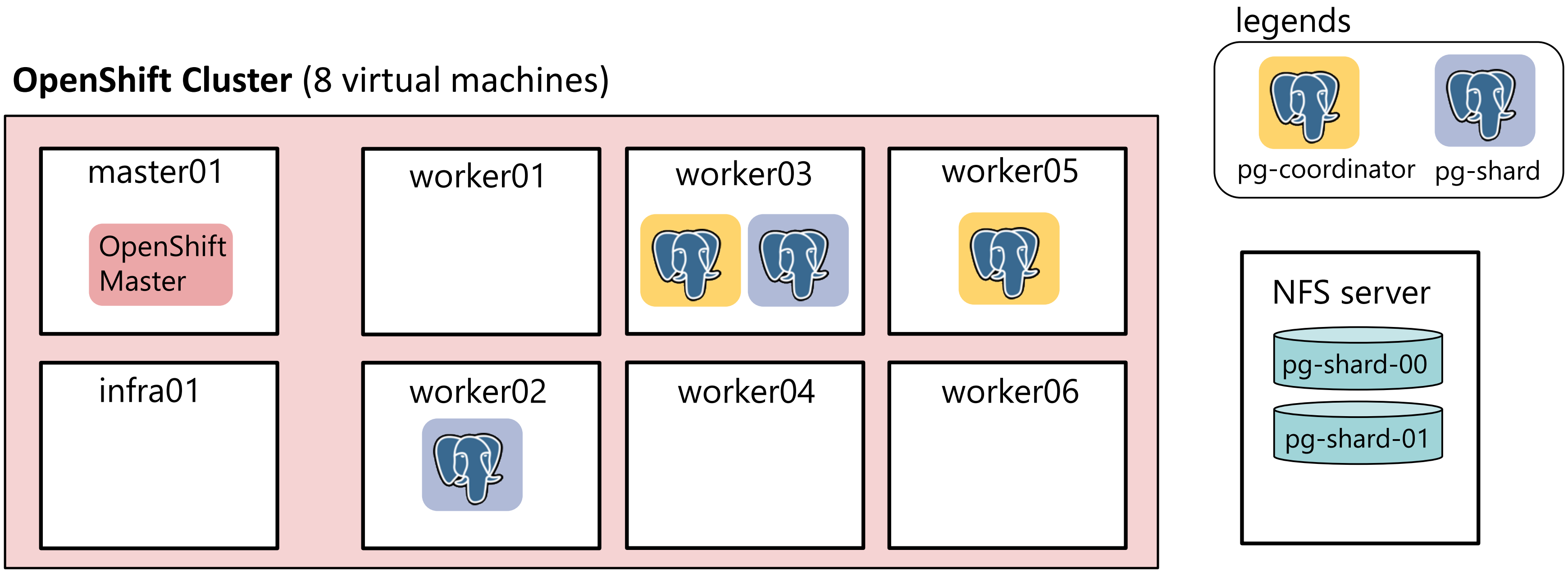
```
$ oc process pg-shard -p PG_PRIMARY_SERVICE_NAME=pg-shard-00 -p
PGDATA_PATH_OVERRIDE=pg-shard-00 | oc create -f -
$ oc process pg-shard -p PG_PRIMARY_SERVICE_NAME=pg-shard-01 -p
PGDATA_PATH_OVERRIDE=pg-shard-01 | oc create -f -
```



3. Multi-Master PostgreSQL Cluster on OpenShift

Check Physical cluster state.

```
$ oc get pod -o wide
NAME                       READY   STATUS    RESTARTS   AGE   IP          NODE
pg-coordinator-1-drtsv     1/1     Running   0          4m    10.131.0.37 worker05
pg-coordinator-1-rmkb8     1/1     Running   0          4m    10.130.0.50 worker03
pg-shard-00-1-qqwtr        1/1     Running   0          4m    10.129.0.68 worker02
pg-shard-01-1-ht4jn        1/1     Running   0          3m    10.130.0.51 worker03
```

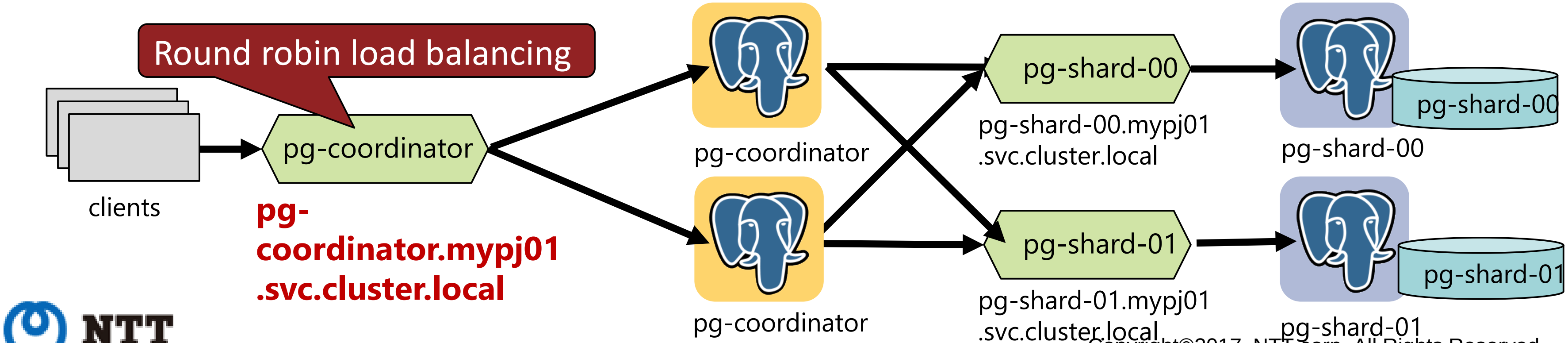


3. Multi-Master PostgreSQL Cluster on OpenShift

Execute SQL to the cluster.

The same schema and data as "Sharding in Build" in the morning session.

```
$ psql -h pg-coordinator.mypj01.svc.cluster.local -U postgres -d postgres ¥
   -c "explain analyze select * from flight_bookings";

                            QUERY PLAN
------------------------------------------------------------------------------------
Append  (cost=100.00..254.09 rows=974 width=144) (actual time=1.945..40.289 rows=7499
loops=1)
  ->  Foreign Scan on flight_bookings1  (cost=...) (actual time=...)
  ->  Foreign Scan on flight_bookings0  (cost=...) (actual time=...)
Planning time: 1.930 ms
Execution time: 60.319 ms
(5 rows)
```


Round robin load balancing
clients
pg-coordinator
pg-coordinator.mypj01.svc.cluster.local
pg-coordinator
pg-coordinator
pg-shard-00
pg-shard-00.mypj01.svc.cluster.local
pg-shard-00
pg-shard-00
pg-shard-01
pg-shard-01.mypj01.svc.cluster.local
pg-shard-01
pg-shard-01

3. Multi-Master PostgreSQL Cluster on OpenShift

Try worker03 to crash.

Execute this command to generate kernel panic on worker 03.

```
$ sudo sh -c "echo c > /proc/sysrq-trigger"
```

Worker 03 status changes “NotReady”.

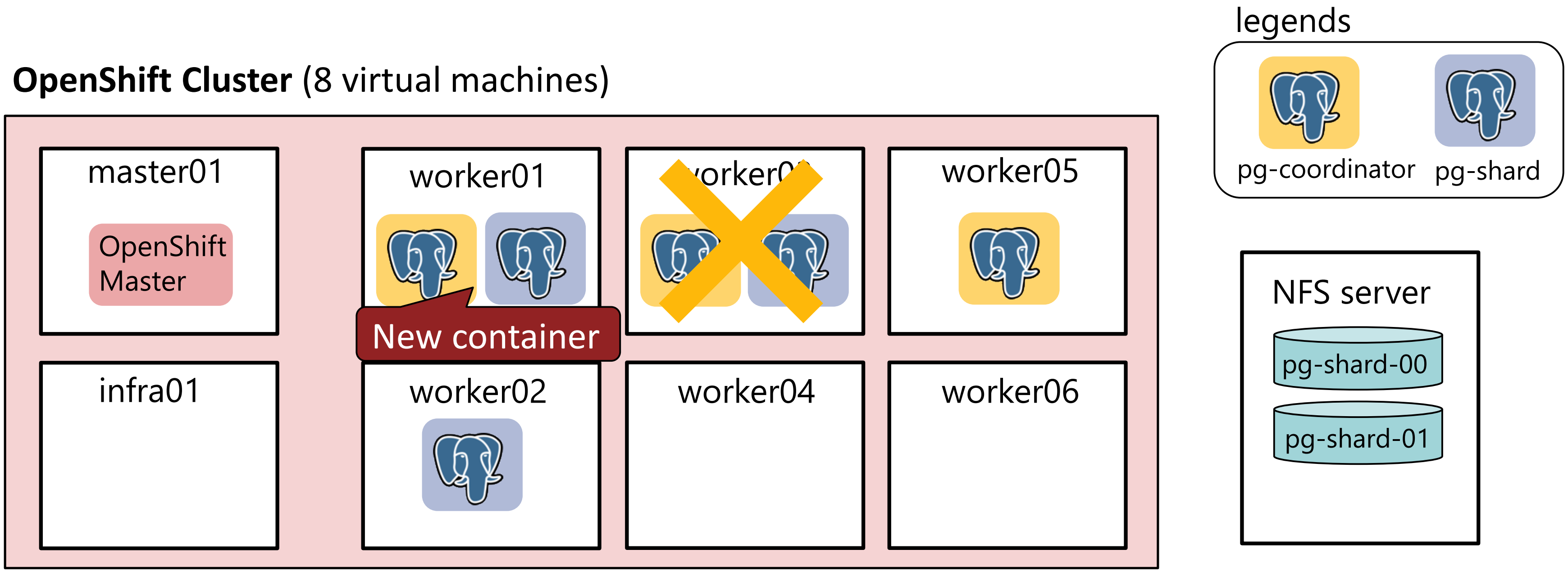
```
$ oc get node
NAME        STATUS                      AGE       VERSION
infra01     Ready                       26d       v1.7.6+a08f5eeb62
master01    Ready,SchedulingDisabled    26d       v1.7.6+a08f5eeb62
worker01    Ready                       26d       v1.7.6+a08f5eeb62
worker02    Ready                       26d       v1.7.6+a08f5eeb62
worker03    NotReady                    26d       v1.7.6+a08f5eeb62
worker04    Ready                       26d       v1.7.6+a08f5eeb62
worker05    Ready                       26d       v1.7.6+a08f5eeb62
worker06    Ready                       26d       v1.7.6+a08f5eeb62
```

3． Multi-Master PostgreSQL Cluster on OpenShift

Containers are re-creating.

```
$ oc get pod -o wide
NAME                      READY   STATUS              RESTARTS   AGE   IP            NODE
pg-coordinator-1-drtsv    1/1     Running             0          8m    10.131.0.37   worker05
pg-coordinator-1-rmkb8    1/1     Terminating         0          8m    10.130.0.50   worker03
pg-coordinator-1-wk4wx    0/1     ContainerCreating   0          2s    <none>        worker01
pg-shard-00-1-qqwtr       1/1     Running             0          8m    10.129.0.68   worker02
pg-shard-01-1-ht4jn       1/1     Terminating         0          8m    10.130.0.51   worker03
pg-shard-01-1-ztdn4       0/1     ContainerCreating   0          2s    <none>        worker01
```

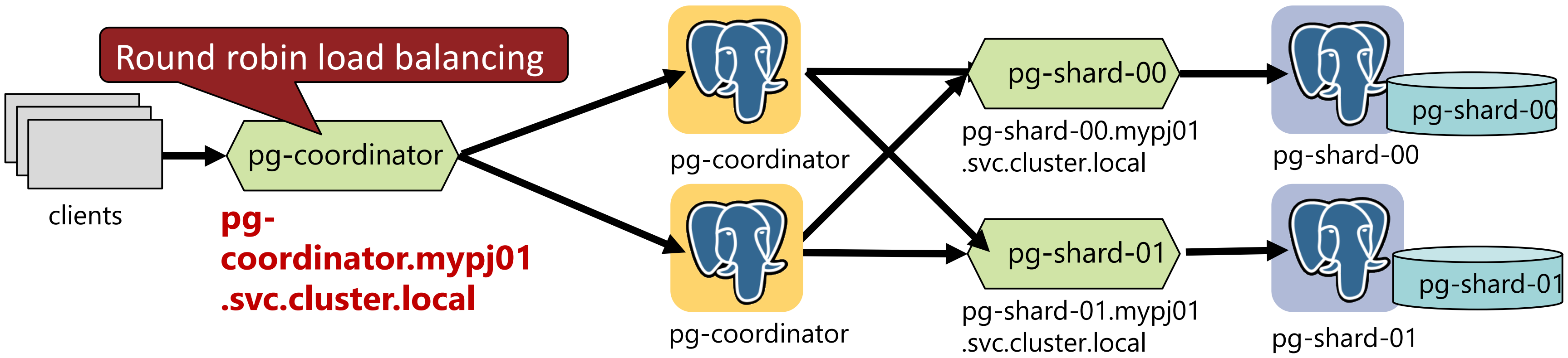
OpenShift Cluster (8 virtual machines)



3． Multi-Master PostgreSQL Cluster on OpenShift

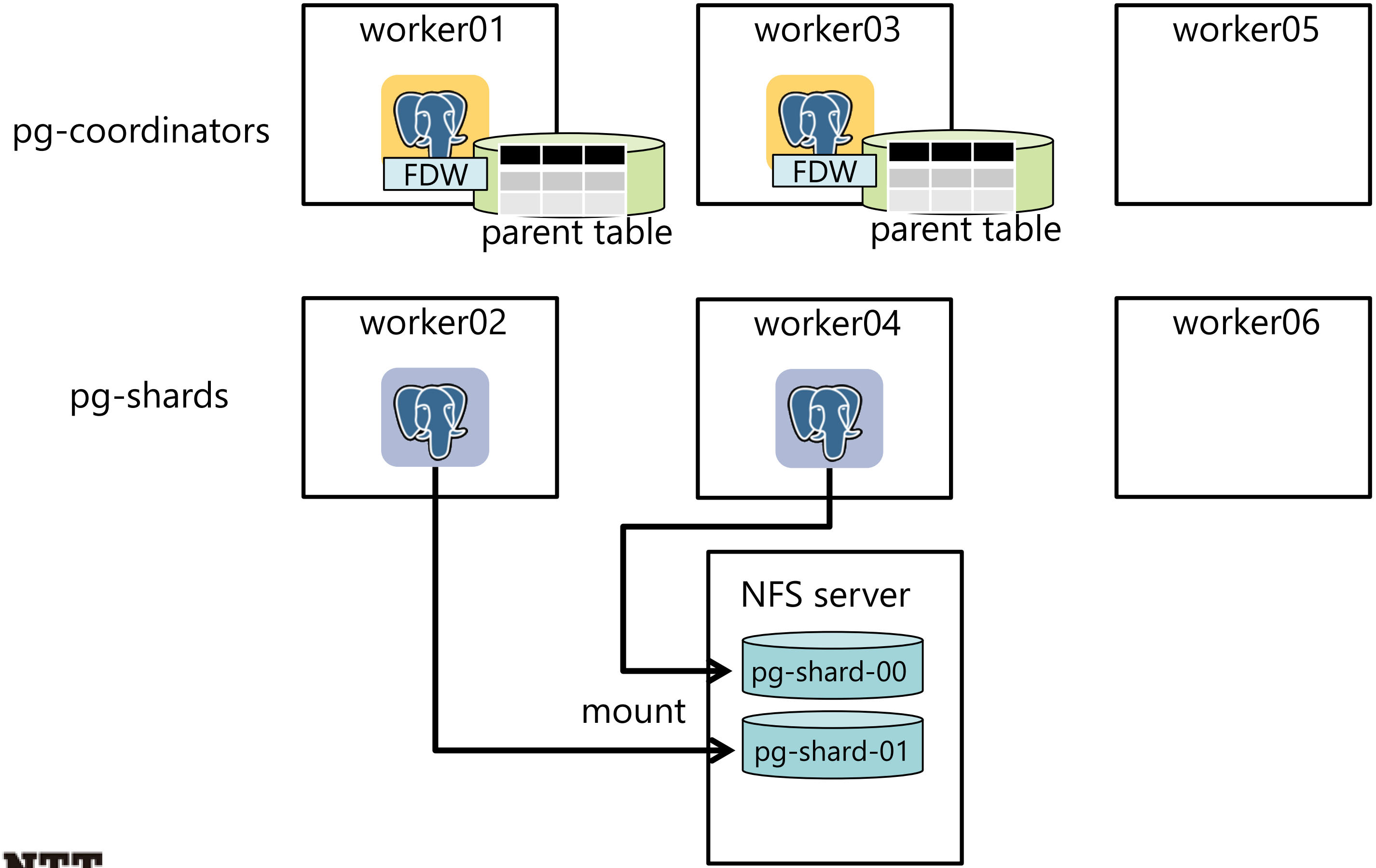
Re-execute SQL.

```
$ psql -h pg-coordinator.mypj01.svc.cluster.local -U postgres -d postgres ¥
  -c "explain analyze select * from flight_bookings";
                                        QUERY PLAN
-----------------------------------------------------------------------------------
Append  (cost=100.00..254.09 rows=974 width=144) (actual time=15.531..63.278 rows=7499
loops=1)
  ->  Foreign Scan on flight_bookings1  (cost=...) (actual time=...)
  ->  Foreign Scan on flight_bookings0  (cost=...) (actual time=...)
Planning time: 1.716 ms
Execution time: 116.989 ms
(5 rows)
```


Round robin load balancing
clients
pg-coordinator
pg-coordinator.mypj01.svc.cluster.local
pg-coordinator
pg-coordinator
pg-shard-00
pg-shard-00.mypj01.svc.cluster.local
pg-shard-01
pg-shard-01.mypj01.svc.cluster.local
pg-shard-00
pg-shard-00
pg-shard-01
pg-shard-01

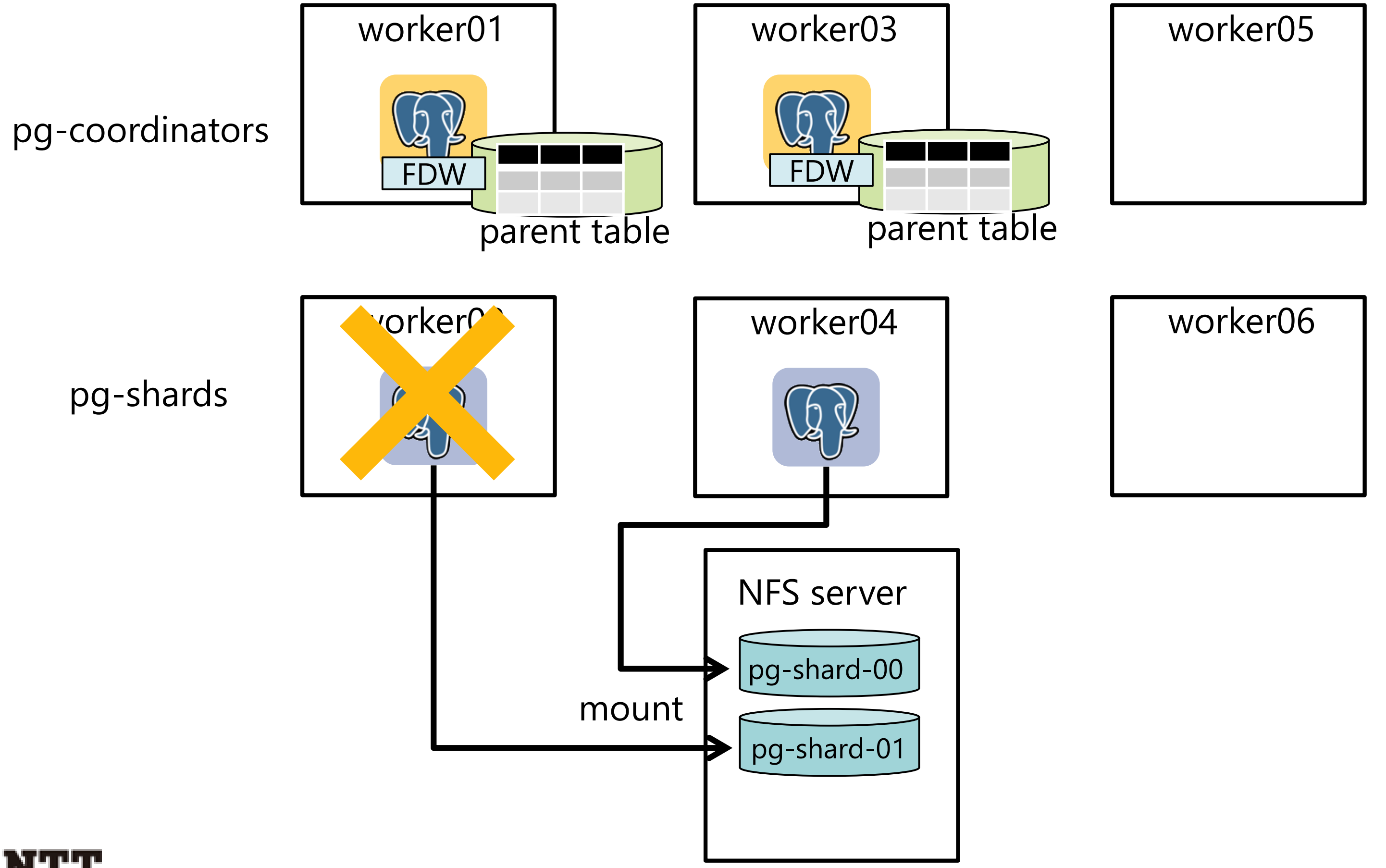
3．Multi-Master PostgreSQL Cluster on OpenShift

pg-coordinators don't have persistent storage. So they lost the cluster configurations if restarted.



3. Multi-Master PostgreSQL Cluster on OpenShift

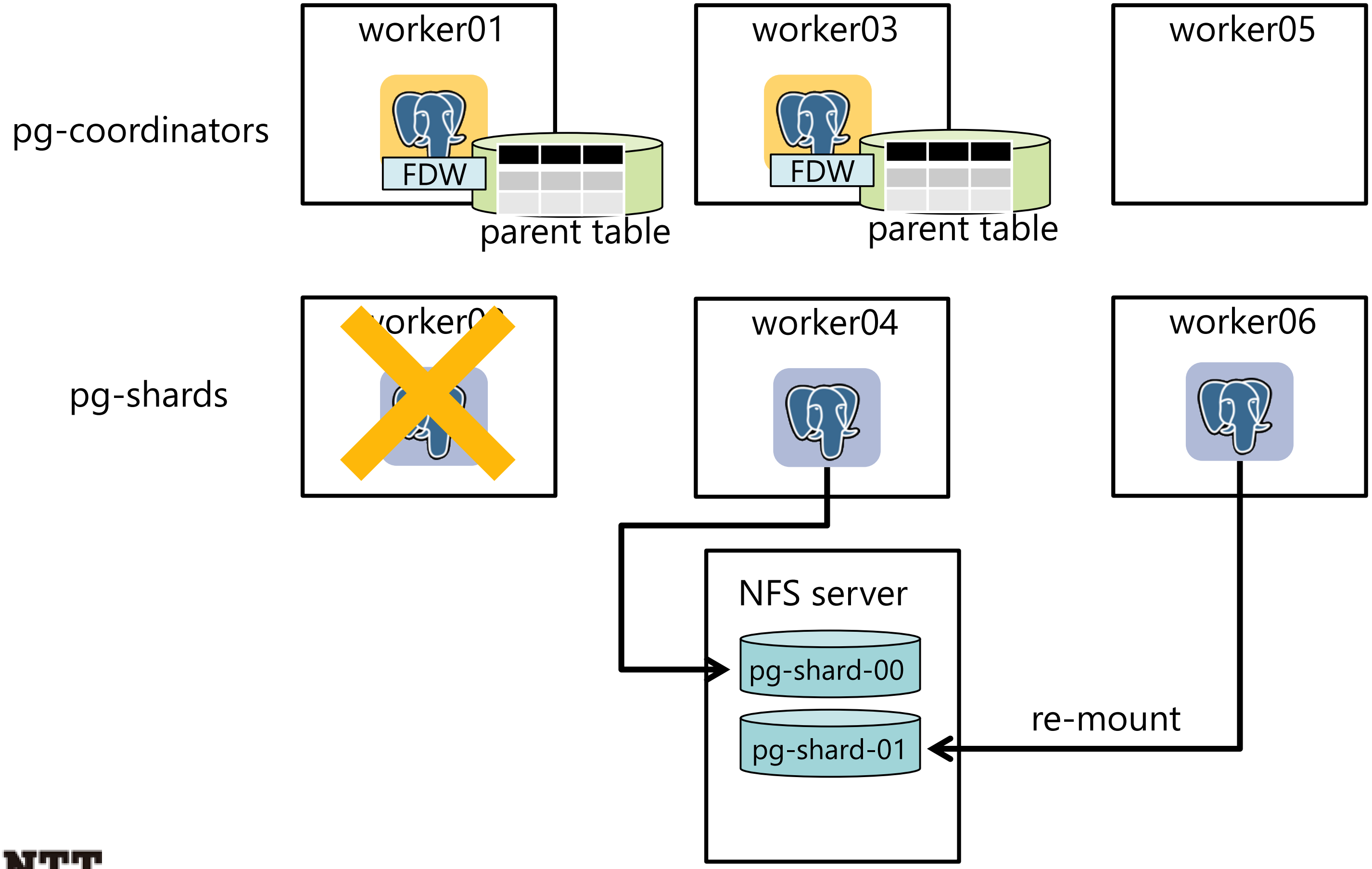
pg-coordinators don't have persistent storage. So they lost the cluster configurations if restarted.



Copyright©2017 NTT corp. All Rights Reserved.

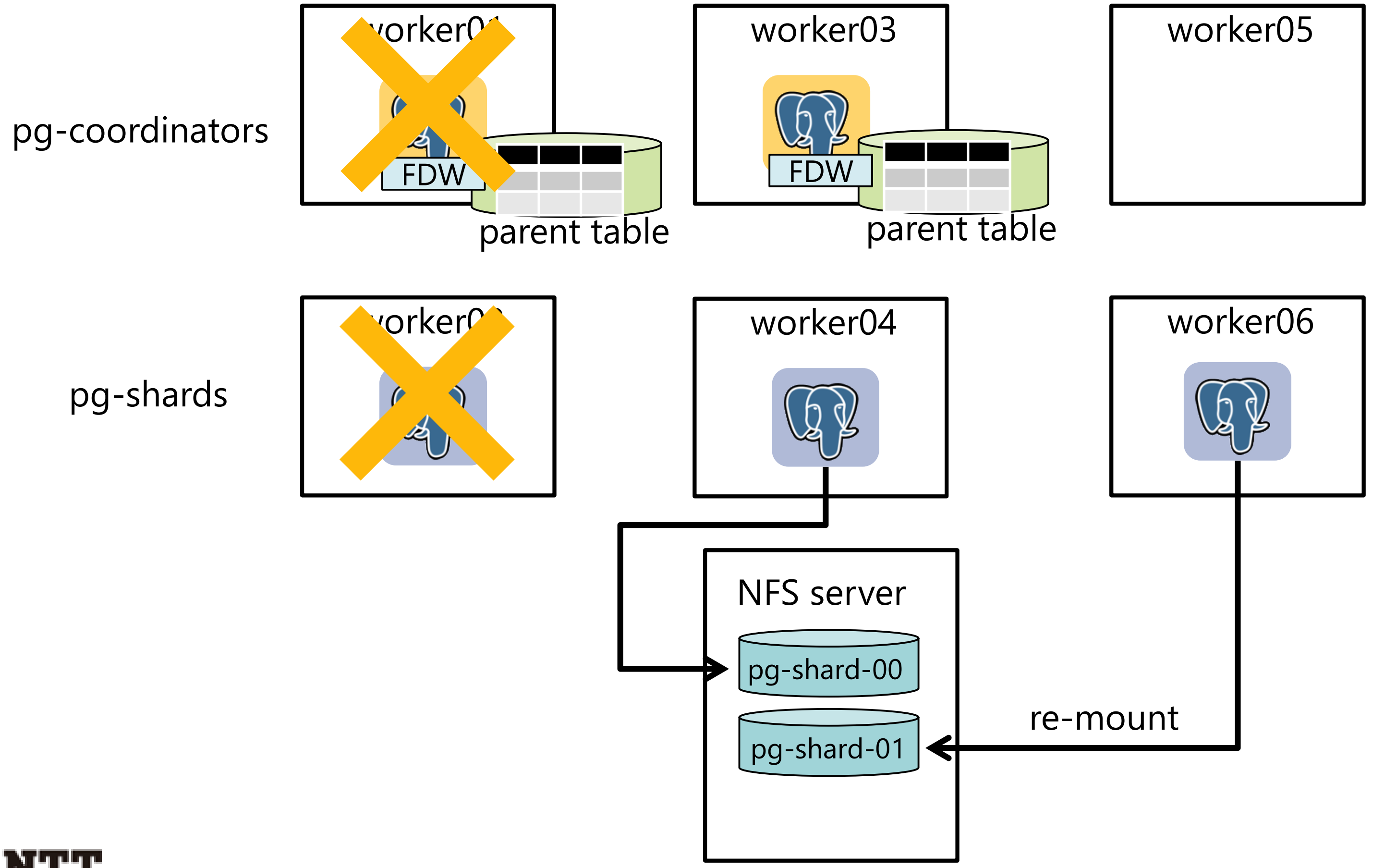
3. Multi-Master PostgreSQL Cluster on OpenShift

pg-coordinators don't have persistent storage. So they lost the cluster configurations if restarted.



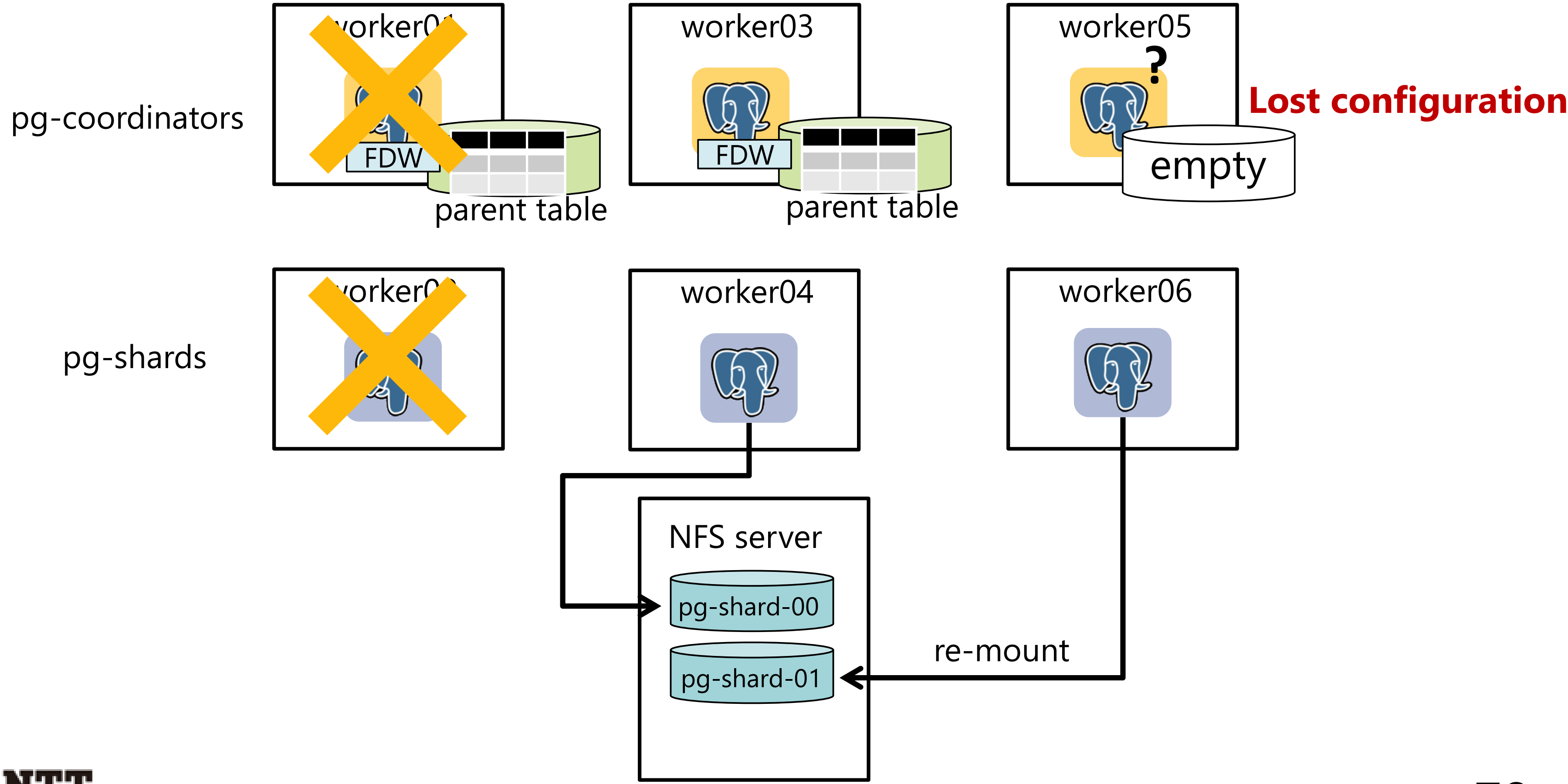
3. Multi-Master PostgreSQL Cluster on OpenShift

pg-coordinators don't have persistent storage. So they lost the cluster configurations if restarted.



3. Multi-Master PostgreSQL Cluster on OpenShift

pg-coordinators don't have persistent storage. So they lost the cluster configurations if restarted.

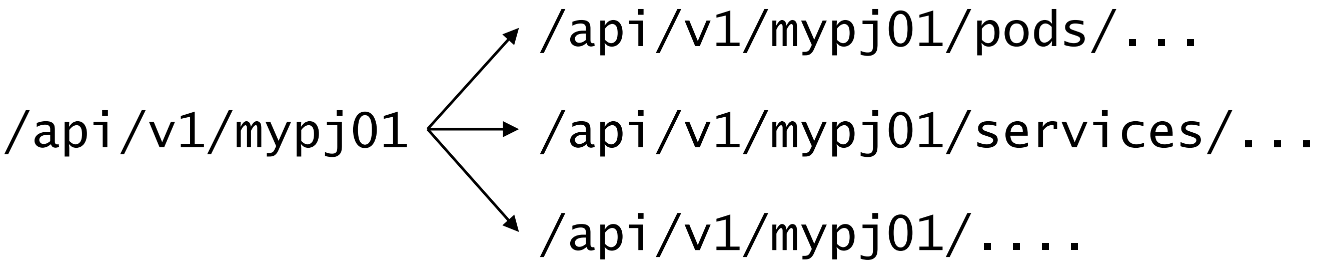


3．Multi-Master PostgreSQL Cluster on OpenShift

Custom Resource Definitions (CRD)

We try to use **CDR(CustomResourceDefinitions)** for sharing the cluster configurations.

- A image of the HTTP API space of kubernetes



/api/v1/mypj01

/api/v1/mypj01/pods/...

/api/v1/mypj01/services/...

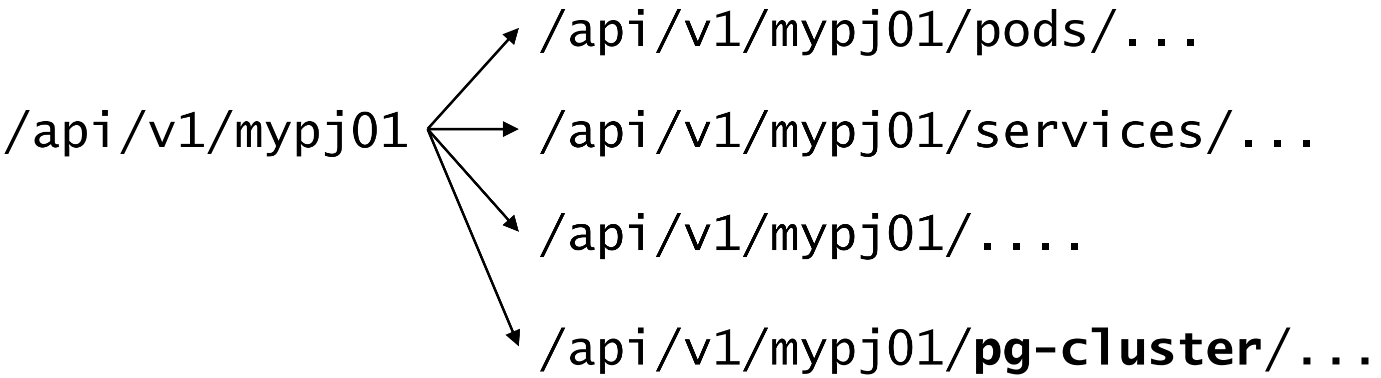
/api/v1/mypj01/....

3．Multi-Master PostgreSQL Cluster on OpenShift

Custom Resource Definitions (CRD)

We try to use **CDR(CustomResourceDefinitions)** for sharing the cluster configurations.

- A image of the HTTP API space of kubernetes



/api/v1/mypj01

/api/v1/mypj01/pods/...

/api/v1/mypj01/services/...

/api/v1/mypj01/....

/api/v1/mypj01/**pg-cluster**/...

-> CDR can extend the kubernetes core API.

3． Multi-Master PostgreSQL Cluster on OpenShift

Write the CDR manifest file.

```
$ cat pg-cluster-shard-information.yml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
 name: shard-informations.pg-cluster.example.com
spec:
 group: pg-cluster.example.com
 version: v1
 scope: Namespaced
 names:
   plural: shard-information
   singular: shard-information
   kind: Shard-Info
   shortNames:
   - si
```

```
$ oc get shard-information
No resources found.
```

This CRD can be access by “oc get shard-information“ or “oc get si“

Write the shard-information manifest file.

```
$ cat defines/sharding_info_coordinator.yml
apiVersion: "pg-cluster.example.com/v1"
kind: Shard-Info
metadata:
 name: multi-master-demo-pgconfasia
spec:
 coordinator:
   "create extension if not exists postgres_fdw;

    create server shard0 foreign data wrapper postgres_fdw options (host 'pg-shard-00.mypj01.svc.cluster.local', dbname 'postgres', port '5432');
    create server shard1 foreign data wrapper postgres_fdw options (host 'pg-shard-01.mypj01.svc.cluster.local', dbname 'postgres', port '5432');

    create user mapping for postgres server shard0 OPTIONS (user 'postgres', password 'password');
    create user mapping for postgres server shard1 OPTIONS (user 'postgres', password 'password');

    create table hotel_bookings (id serial, user_id int, booked_at timestamp, city_name text, continent text, flight_id int) partition by list (continent);
    create table flight_bookings (id serial, user_id int, booked_at timestamp, from_city text, from_continent text, to_city text, to_continent text) partition
by list (to_continent);
    create table users (id serial, name text, age int);

    create foreign table flight_bookings0 partition of flight_bookings for values in ('Asia', 'Oceania') server shard0;
    create foreign table hotel_bookings0 partition of hotel_bookings for values  in ('Asia', 'Oceania') server shard0;

    create foreign table flight_bookings1 partition of flight_bookings for values in ('Europe', 'Africa') server shard1;
    create foreign table hotel_bookings1 partition of hotel_bookings for  values in ('Europe', 'Africa') server shard1;
```

3. Multi-Master PostgreSQL Cluster on OpenShift

Write the shard-information manifest file.

```
$ cat defines/sharding_info_coordinator.yml
apiVersion: "pg-cluster.example.com/v1"
kind: Shard-Info
metadata:
 name: multi-master-demo-pgconfasia
spec:
 coordinator:
   "create extension if not exists postgres_fdw;

    create server shard0 foreign data wrapper postgres_fdw options (host 'pg-shard-00.mypj01.svc.cluster.local', dbname 'postgres', port '5432');
    create server shard1 foreign data wrapper postgres_fdw options (host 'pg-shard-01.mypj01.svc.cluster.local', dbname 'postgres', port '5432');

    create user mapping for postgres server shard0 OPTIONS (user 'postgres', password 'password');
    create user mapping for postgres server shard1 OPTIONS (user 'postgres', password 'password');

    create table hotel_bookings (id serial, user_id int, booked_at timestamp, city_name text, continent text, flight_id int) partition by list (continent);
    create table flight_bookings (id serial, user_id int, booked_at timestamp, from_city text, from_continent text, to_city text, to_continent text) partition
by list (to_continent);
    create table users (id serial, name text, age int);

    create foreign table flight_bookings0 partition of flight_bookings for values in ('Asia', 'Oceania') server shard0;
    create foreign table hotel_bookings0 partition of hotel_bookings for values  in ('Asia', 'Oceania') server shard0;

    create foreign table flight_bookings1 partition of flight_bookings for values in ('Europe', 'Africa') server shard1;
    create foreign table hotel_bookings1 partition of hotel_bookings for  values in ('Europe', 'Africa') server shard1;
```

There are SQLs for setting up to pg-coordinator.

3． Multi-Master PostgreSQL Cluster on OpenShift

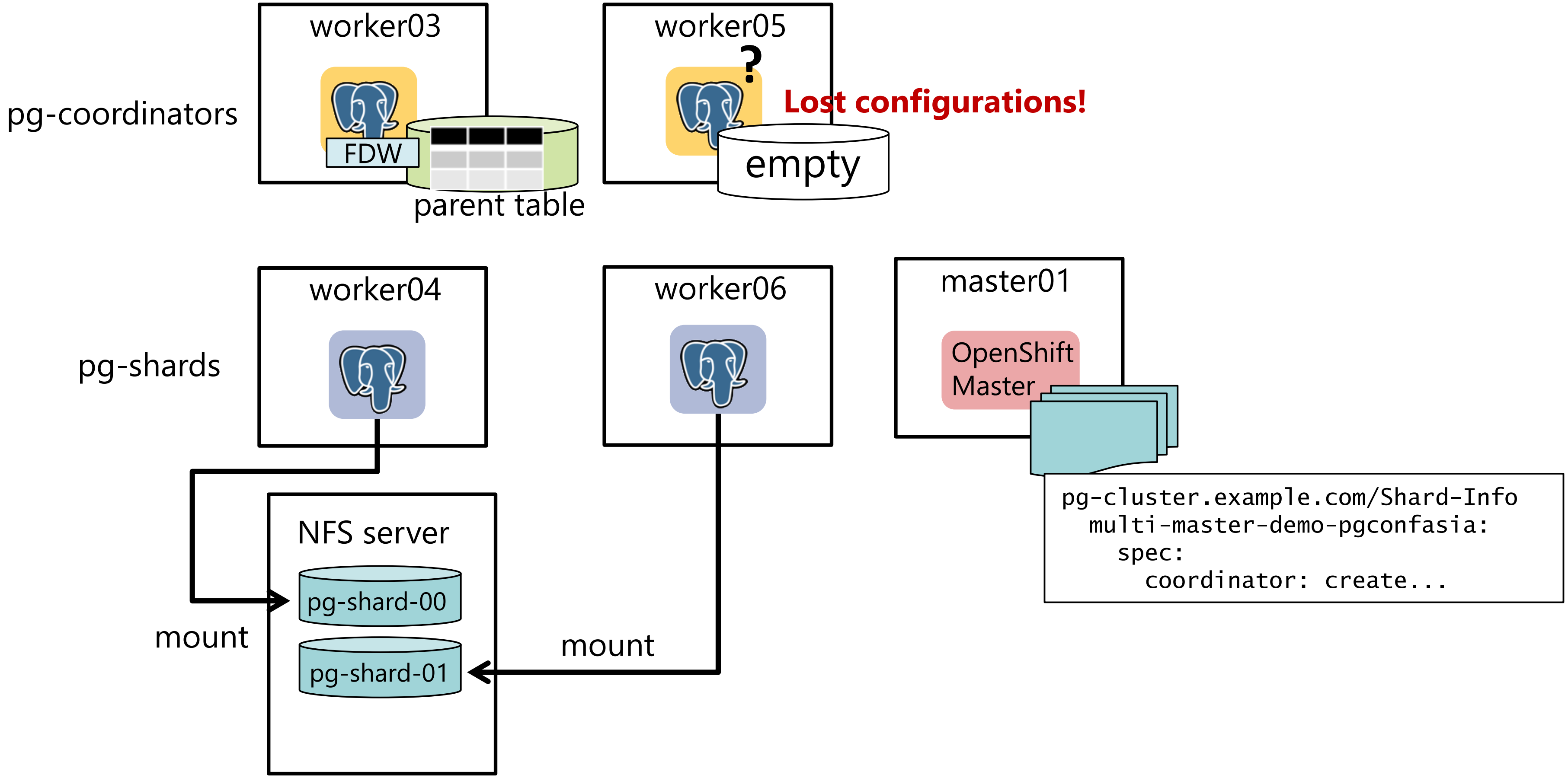
We can get these SQLs through “oc” command.

```
$ oc get si multi-master-demo-pgconfasia -o json | jq -r '.spec.coordinator'
   "create extension if not exists postgres_fdw;

    create server shard0 foreign data wrapper postgres_fdw options (host 'pg-shard-00.mypj01.svc.cluster.local', dbname 'postgres', port '5432');
    create server shard1 foreign data wrapper postgres_fdw options (host 'pg-shard-01.mypj01.svc.cluster.local', dbname 'postgres', port '5432');

    create user mapping for postgres server shard0 OPTIONS (user 'postgres', password 'password');
    create user mapping for postgres server shard1 OPTIONS (user 'postgres', password 'password');

    create table hotel_bookings (id serial, user_id int, booked_at timestamp, city_name text, continent text, flight_id int) partition by list (continent);
    create table flight_bookings (id serial, user_id int, booked_at timestamp, from_city text, from_continent text, to_city text, to_continent text) partition
by list (to_continent);
    create table users (id serial, name text, age int);

    create foreign table flight_bookings0 partition of flight_bookings for values in ('Asia', 'Oceania') server shard0;
    create foreign table hotel_bookings0 partition of hotel_bookings for values  in ('Asia', 'Oceania') server shard0;

    create foreign table flight_bookings1 partition of flight_bookings for values in ('Europe', 'Africa') server shard1;
    create foreign table hotel_bookings1 partition of hotel_bookings for  values in ('Europe', 'Africa') server shard1;
```

pg-coordinator can also take out these SQLs from the Shard-Info CDR.

3． Multi-Master PostgreSQL Cluster on OpenShift

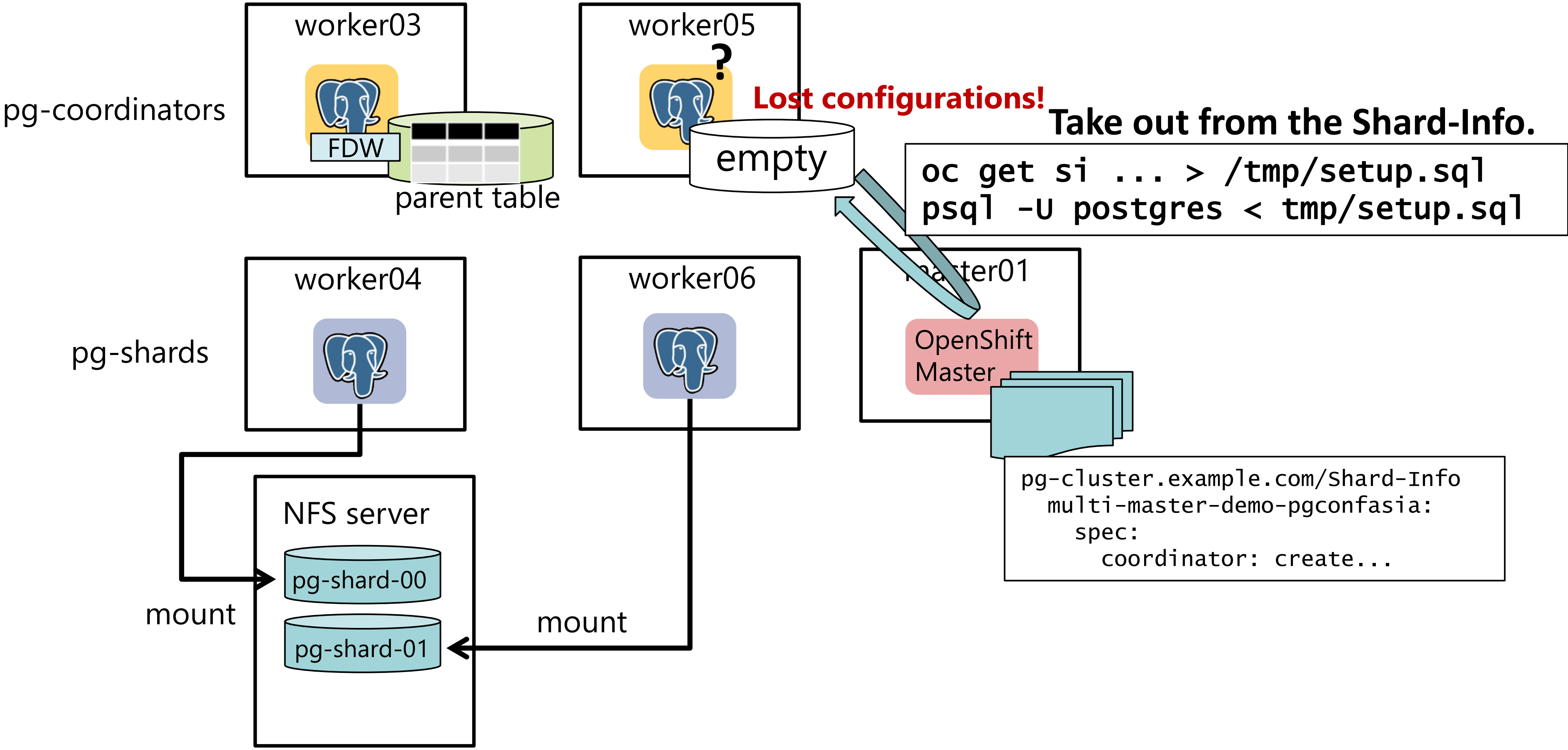
pg-coordinator takes out the setting SQLs from the Shard-Info CDR when it is started.



```
pg-cluster.example.com/Shard-Info
  multi-master-demo-pgconfasia:
    spec:
      coordinator: create...
```

3. Multi-Master PostgreSQL Cluster on OpenShift

pg-coordinator takes out the setting SQLs from the Shard-Info CDR when it is started.



Lost configurations!

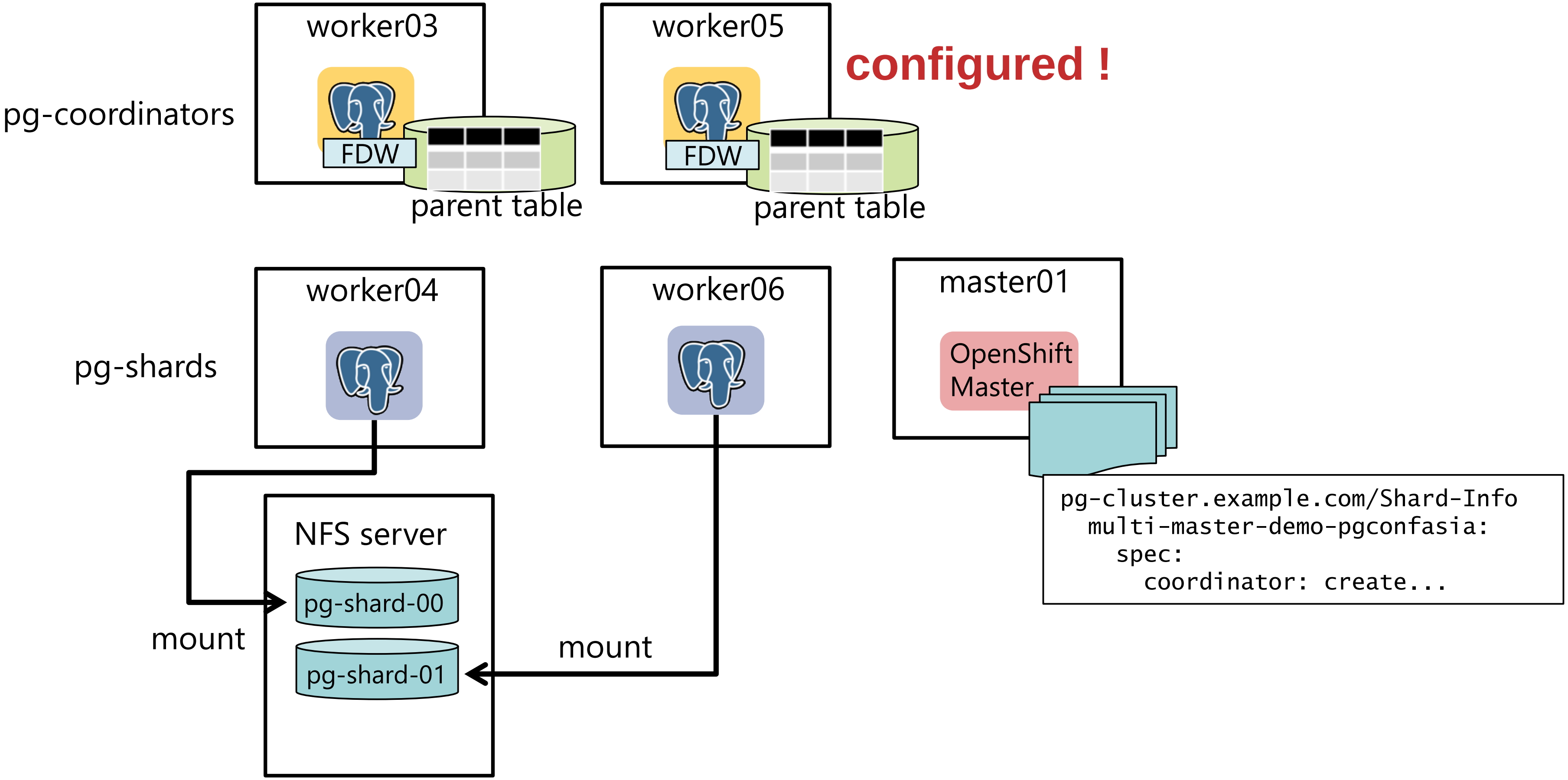
Take out from the Shard-Info.

```
oc get si ... > /tmp/setup.sql
psql -U postgres < tmp/setup.sql
```

```
pg-cluster.example.com/Shard-Info
  multi-master-demo-pgconfasia:
    spec:
      coordinator: create...
```

3． Multi-Master PostgreSQL Cluster on OpenShift

pg-coordinator takes out the setting SQLs from the Shard-Info CDR when it is started.



```
pg-cluster.example.com/Shard-Info
  multi-master-demo-pgconfasia:
    spec:
      coordinator: create...
```

3． Multi-Master PostgreSQL Cluster on OpenShift

Let’s scale out pg-coordinator.

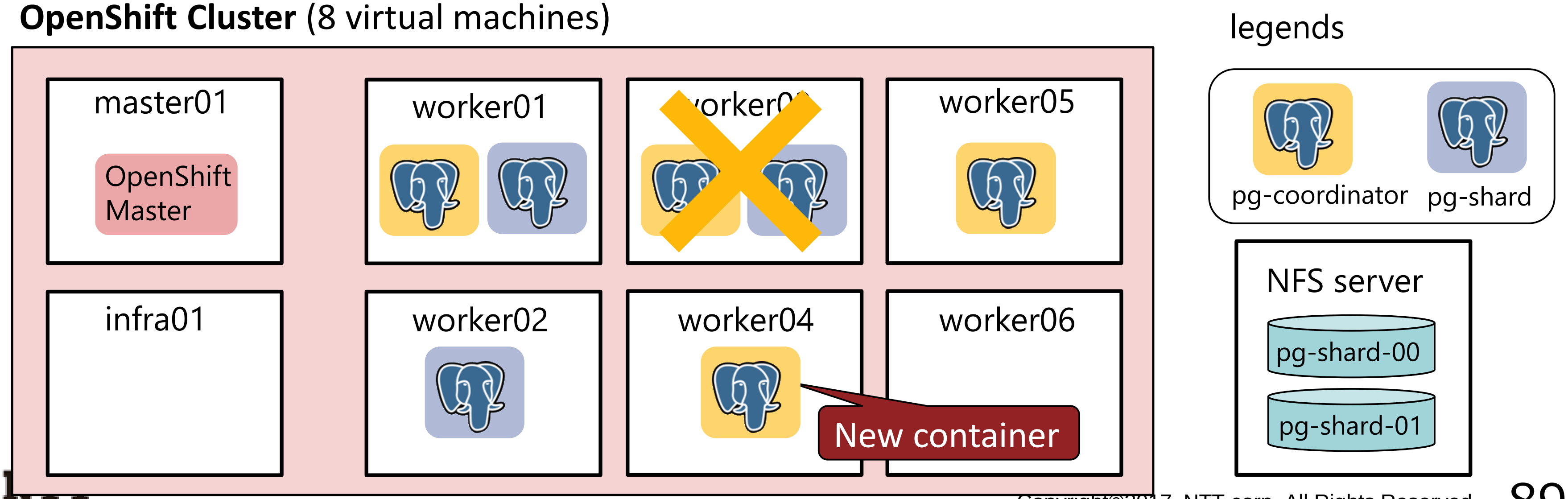
- Set “replicas” 2 -> 3

```
$ oc scale dc pg-coordinator --replicas=3
deploymentconfig "pg-coordinator" scaled
```

- Create Service and DeploymentConfig from the Template.

```
$ oc get pod -o wide
NAME                    READY    STATUS    RESTARTS   AGE   IP            NODE
pg-coordinator-1-drtsv  1/1      Running   0          20m   10.131.0.37   worker05
pg-coordinator-1-wk4wx  0/1      Running   0          11m   10.128.0.76   worker01
pg-coordinator-1-87ddq  0/1      Running   0          1m    10.129.2.45   worker04
pg-shard-00-1-qqwtr     1/1      Running   0          20m   10.129.0.68   worker02
pg-shard-01-1-ztdn4     0/1      Running   0          11m   10.128.0.77   worker01
```

OpenShift Cluster (8 virtual machines)



4． Conclusion

- PostgreSQL on Kubernetes is easy to run PostgreSQL even if Multi-Master PostgreSQL Cluster.
  - including HA, monitoring etc.
  - easy to scale out.
- Custom Resource Definitions is useful for sharing the cluster configurations.

but you have to consider ...

➢ Persistent storage.

➢ In case of server/container failure.

➢ Kubernetes cluster management.

Please try PostgreSQL on kubernetes! You too!
And share the knowledge.

Future Plan
We try to use Red Hat Ceph Storage as a distributed block storage instead of NFS. This can make the storage layer scalable! And evaluate the Performance.