Technical Guide

Troubleshooting Kubernetes applications

September 2021

SAMSUNG SDS

Contents

1. Overview

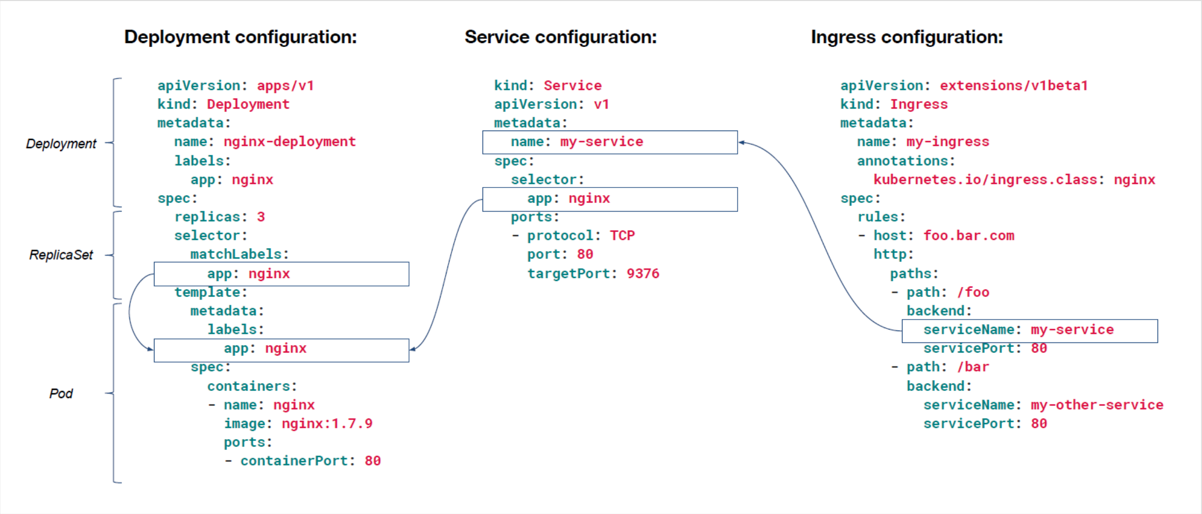
The purpose of this document is to describe how to troubleshoot issues while deploying workload on a Kubernetes cluster or operating the workload.

The most commonly used web application type will be introduced in the document. You can also refer to this document to troubleshoot SDS Cloud products of Kubernetes Engine and Kubernetes Apps, as well as vanilla Kubernetes and public managed Kubernetes including GKE, AKS, and EKS.

2. Typical application structure

Kubernetes offers various mechanisms such as hostNetwork, nodePort, and type: LoadBalancer to enable external access for web applications, but we will deal with the workload of the Ingress method, which is easy to access the http/https-based domain call method.

Here is an example Kubernetes resource combination for deploying an Nginx web application as a Kubernetes workload.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: my-service
          servicePort: 80
      - path: /bar
        backend:
          serviceName: my-other-service
          servicePort: 80
```

Figure 1. Deployment strategies on Kubernetes, CNCF

A variety of other resources such as persistent volume or network policy may exist but the example above is the most basic Kubernetes resource using Ingress.

The important thing here is the reference relationship between each resource. You must remember that both deployment and service selectors refer to the label of the pod template while serviceName refers to the name of the service resource in Ingress.

Next, we will look at normal status checks and responses to potential issues after the deployment of each resource.

3. Health check on pods

3.1 Basic commands

Here are some of the kubectl commands needed to check the status of a pod.

The following are the most basic query commands to check the basic pod status, an extended command including IP and assigned node information (-owide), or raw data in the form of a YAML manifest (-oyaml).

```
$ kubectl get pod
$ kubectl get pod -owide
$ kubectl get pod -oyaml
```

The following command is mainly used to check detailed information and events of pods if a problem occurs.

```
$ kubectl describe po <pod-name>
```

The command below can check the log if there is a problem with the running application. If multiple containers are running in the pod, you can select a specific container through the -c option, enable real-time tailing with the -f option and check logs for previous pods using the –previous option.

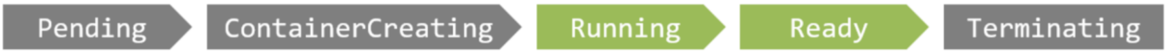
```
$ kubectl logs <pod-name> [-c <container-name>]
$ kubectl logs <pod-name> [-c <container-name>] -f
$ kubectl logs <pod-name> [-c <container-name>] --previous
```

The following is a command that shows the events that occurred in the namespace, which can be sorted by a chronological order with the --sort-by option.

```
$ kubectl get ev --sort-by=.metadata.creationTimestamp
```

3.2 Lifecycle of normal pods

After deploying a pod, the flow of the states until the pod changes to a healthy state has to be tracked. A typical pod has the following lifecycle.



- **Pending**: This status is displayed before determining which node will be scheduled to a pod or if there are no schedulable nodes.
- **ContainerCreating**: This stage is when the container runtime of a node creates

a container after a pod is scheduled on a specific node. It can be displayed in the process of mounting related resources, such as image pulling, ConfigMaps, and persistent volume.

- **Running**: The process in the pod is running normally.
- **Ready**: After the Running status, the pod is ready to provide service as per the readiness probe set in the pod.
- **Terminating**: This status may appear in the process of cleaning up a pod from the applicable node when the pod has been deleted.

You can check changes in the pod lifecycle by using the –w (watch) option in the kubectl get po command below. The most basic columns to check the normal status of the pod among these are READY and STATUS.

```
$ kubectl get po -owide -w
NAME                     READY   STATUS              AGE   IP           NODE
nginx-5d796fc999-qkgpp   0/1     Pending             0s    <none>       <none>
nginx-5d796fc999-qkgpp   0/1     Pending             0s    <none>       node1
nginx-5d796fc999-qkgpp   0/1     ContainerCreating   1s    <none>       node1
nginx-5d796fc999-qkgpp   0/1     Running             3s    10.44.0.2    node1
nginx-5d796fc999-qkgpp   1/1     Running             9s    10.44.0.2    node1
```

Next, we will look at the causes of and responses for each case of abnormal pod status.

3.3 If Pending state persists

Sometimes a pod's state stays in Pending for a long time.

```
$ kubectl get po
NAME                    READY    STATUS     RESTARTS    AGE
nginx-6dcd8d4dff-njsrh   0/1     Pending    0           10m
```

In this case, Event messages need to be checked first using the kubectl describe pod command.

```
$ kubectl describe po <pod-name>
Events:
  Type     Reason              Age               From
  Message
  ----     ------              ----              ----                -----
  Warning  FailedScheduling    14s (x2 over 14s)  default-scheduler  0/3 nodes are available: 3
Insufficient cpu/memory.
```

In the above case, the cluster is running out of resources. You should check the Allocated resources: assigned to each node through the kubectl describe node command, and if necessary, increase the cluster node resources or reduce the pod resources.

```
$ kubectl describe po <pod-name>
Events:
  Type     Reason              Age              From               Message
  ----     ------              ----             ----               -------
  Warning  FailedScheduling  5s (x2 over 5s)  default-scheduler  0/3 nodes are available: 3
node(s) didn't match node selector.
```

In this case, there is no node matching the nodeSelector/nodeAffinity set in the pod. You can normally schedule a pod by modifying the nodeSelector/nodeAffinity of the pod to the matching node information or adding a label to the node.

```
$ kubectl describe po <pod-name>
Events:
  Type     Reason              Age              From               Message
  ----     ------              ----             ----               -------
  Warning  FailedScheduling  6s (x2 over 6s)  default-scheduler  0/3 nodes are available: 3
node(s) were unschedulable.
```

In this case, the node is in an unschedulable state, as all nodes are cordoned or in NotReady state to cause the taint to be hanging. You must either uncordon the node or act on the node in NotReady state.

```
$ kubectl describe po <pod-name>
Events:
  Type     Reason              Age              From               Message
  ----     ------              ----             ----               -------
  Warning  FailedScheduling   53s (x2 over 60s)   default-scheduler   error while running
"VolumeBinding" filter plugin for pod "nginx-9d8d9896-k7gzj": pod has unbound immediate
PersistentVolumeClaims
```

You need to make sure that the PVC (PersistentVolumeClaim) is not in Pending state.

```
$ kubectl get pvc
NAME          STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
nginx-pvc    Pending                                     standard
```

When using a static PV (PersistentVolume), the PV in the Available state is normally created first, and then you need to check whether the PV name properly matches to the volumeName of the PVC.

If you are using dynamic PV, you need to check if there is a StorageClass created with the storageClassName set in the PVC and then check if the Volume Provisioner is working properly.

```
$ kubectl describe po <pod-name>
Events:
  Type     Reason      Age    From                   Message
  ----     ------      ----   ----                   -------
  Normal   Scheduled   21s    default-scheduler   Successfully assigned default/nginx-5d796fc999-
6ksbj to node1
$ kubectl get po -owide
NAME                          READY    STATUS      RESTARTS     AGE         NODE       nginx-
5d796fc999-6ksbj    0/1       Pending     0              37s         node1
```

For cases that only the event indicating a normal assignment of the pod to the node appears while the pod remains Pending, you need to check if the kubelet of the node that displays when you use –owide is working properly and take action accordingly.

```
$ kubectl describe po <pod-name>
Events: <none>
$ kubectl get po –n kube-system
NAME                              READY    STATUS                  RESTARTS    AGE
kube-scheduler-master             0/1      CrashLoopBackOff    9             16m
```

If no message is recorded in the event, kube-scheduler is not properly scheduling and therefore measures need to be taken.

3.4 When ContainerCreating state persists

Below is what happens when a pod stays in the ContainerCreating state.

```
$ kubectl get po
NAME                       READY    STATUS                  RESTARTS    AGE
nginx-6dcd8d4dff-njsrh    0/1      ContainerCreating    0             1m
```

Even in this case, you must first check the Event message using the kubectl describe pod command.

```
$ kubectl describe po <pod-name>
Events:
  Type     Reason      Age          From                   Message
  ----     ------      ----         ----                   -------
  Normal   Scheduled   <unknown>   default-scheduler   Successfully assigned default/nginx-
58cd5b85f7-cmb6c to node1
  Normal   Pulling     12s          kubelet, node1       Pulling image "nginx:1.13.9"
```

If the Event message is stuck at the pulling image state, the images are still being pulled and you need to wait a little longer. Pulling may take long if the image capacity is large or the network performance is low, and the issue will be resolved naturally when image pulling is completed.

```
$ kubectl describe po <pod-name>
Events:
  Type     Reason       Age           From                  Message
  ----     ------       ----          ----                  -------
  Warning  FailedMount  5s (x6 over 21s)  kubelet, node2    MountVolume.SetUp failed for
volume "config-volume" : configmap/secret "nginx-cm" not found
```

The case below is when the ConfigMap/Secret used by the pod is not created or an incorrect name is specified. You need to make sure that the ConfigMap/Secret specified in the volumes field of the pod spec actually exists.

```
$ kubectl describe po <pod-name>
Events:
  Type     Reason       Age           From                  Message
  ----     ------       ----          ----                  -------
  Warning  FailedMount        18s         kubelet, node2    MountVolume.SetUp failed for
volume "pvc-d2def29a-d503-443e-9615-886713983216" : mount failed: exit status 32
```

This occurs when a pod is not mounted to the PV in use. For NFS or Container Storage Interface (CSI) Driver to use the PV, you need to check if nfs-utils, rpcbind, or other clients are installed. In addition, the access control and firewall for the PV in the node need to be checked to enable the mount.

3.5 If ImagePullBackOff status is displayed

This time, we will look at the situation where ImagePullBackOff occurs.

```
$ kubectl get po
NAME                      READY   STATUS              RESTARTS   AGE
nginx-6dcd8d4dff-njsrh   0/1      ImagePullBackOff   0           6s
```

You can check it with the describe command.

```
$ kubectl describe po <pod-name>
Events:
  Type     Reason          Age                 From                  Message
  ----     ------          ----                ----                  -------
  Warning  Failed             23s (x2 over 44s)  kubelet, node1         Failed to pull image
"nginx:invalid-tag": rpc error: code = Unknown desc = Error response from daemon: manifest for
nginx:test not found
```

This is when there is no Image:Tag in the Image Repository. You need to edit the Image:Tag to the correct information in the pod specifications or push the Image:Tag to the Image Repository.

```
$ kubectl describe po <pod-name>
Events:
  Type      Reason         Age                From                 Message
  ----      ------         ----               ----                 -------
  Warning   Failed         9s                      kubelet,  node1        Failed  to  pull  image
"myregistry.io/nginx:1.17.8": rpc error: code = Unknown desc = Error response from daemon: Get
https://myregistry.io/v2/nginx/manifests/1.17.8: no basic auth credentials
```

If you are using a private registry, you may need credential information. In this case, you need to create imagePullSecrets with the correct credential information and add it to the pod spec. If imagePullSecrets is already in use, additional check on the expiration of authority is required.

```
$ kubectl describe po <pod-name>
Events:
  Type      Reason         Age                From                 Message
  ----      ------         ----               ----                 -------
  Warning   Failed     12s    kubelet, node1   Failed to pull image "nginx:1.17.8": rpc error: code =
Unknown desc = Error response from daemon: Get https://registry-1.docker.io/v2/: net/http: request
canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)
```

Above shows a failure of communications with the Image Repository in the node. You need to check if the firewall between Image Repositories and the Image Repository service are properly operating on the node.

3.6 If CrashLoopBackOff and Running repeat periodically

A pod may sometimes repeat the CrashLoopBackOff and Running states.

```
$ kubectl get po -w
NAME                        READY    STATUS              RESTARTS    AGE
test-7f74c45f58-jm629     0/1      CrashLoopBackOff    8              16m
test-7f74c45f58-jm629     0/1      Running            9              16m
```

If this takes place, you need to check the current or previous pod logs using the command below and troubleshoot the application if the log is confirmed.

```
$ kubectl logs po <pod-name> -f
$ kubectl logs po <pod-name> --previous
$ kubectl describe po <pod-name>
 Last State:       Terminated
       Reason:          ContainerCannotRun
       Message:         OCI runtime create failed: container_linux.go:344: starting container process
caused "exec: \"ping\": executable file not found in $PATH": unknown
       Exit Code:      127
```

The most common application issues include DB connection failure, abnormal command use, and application configuration issues.

If you cannot see the current or previous pod logs, check the Last State: details with the describe command.

3.7 If CrashLoopBackOff and Completed repeat periodically

Pods may also repeat CrashLoopBackOff and Completed states.

```
$ kubectl get po -w
NAME                       READY    STATUS              RESTARTS    AGE
test-6fd77b68b9-44hxs     0/1      CrashLoopBackOff    8           16m
test-6fd77b68b9-44hxs     0/1      Completed           9           16m
```

In this case, there is no foreground process inside the container. You must specify a process that can operate in the foreground with CMD or ENTRYPOINT in the Dockerfile or add a commnad to Deploymenet. If it is a batch process, it must be created as a Job or CronJob.

3.8 If status does not change to Ready

The pod is running, but the status does not change to READY.

```
$ kubectl get po
NAME                       READY    STATUS     RESTARTS    AGE
nginx-659484b897-mnq4l      0/1     Running    0           2m14s
```

If this happens, use the describe command.

```
$ kubectl describe po <pod-name>
Events:
  Type     Reason       Age          From                  Message
  ----    ------      ----       ----                 -------
  Warning   Unhealthy   5s (x7 over 65s)   kubelet, node2       Readiness probe failed: HTTP probe
failed with statuscode: 404
```

The Event messages need to be checked to see if the readiness probe has failed, followed by necessary measures. If the application startup time is too long, initailDelaySeconds has to be increased.

3.9 If Terminating state continues

The next case is when the Terminating state persists.

```
$ kubectl get po -owide
NAME                       READY   STATUS        RESTARTS   AGE   NODE
nginx-6dcd8d4dff-njsrh     0/1     Terminating   0          10m   node2
nginx-6dcd8d4dff-fvlmf     1/1     Running       0          43s   node1
```

First, we need to make sure that the node to which the pod is assigned is not in NotReady state. It is also important to note that in NotReady state, pods have different Eviction policies for different types of workloads.

```
$ kubectl get no
NAME     STATUS     ROLES    AGE   VERSION
master   Ready      master   39d   v1.21.2
node1    Ready      worker   39d   v1.21.2
node2    NotReady   worker   39d   v1.21.2
```

- Deployment: Check if it has failed over to another node
- StatefulSet: Not moved to another node (can be moved when forcibly deleted)
- DaemonSet: Not moved to another node

If the node to which the pod is assigned is Ready, the issue may be that PV is unmounted or detach does not work properly during the pod shutdown process, requiring a check on storage connection.
For other reasons, you should see if a zombie (defunct) process has been created on the node.

3.10 If Evicted

This is when the pod is Evicted.

```
$ kubectl get po -owide
NAME                       READY   STATUS    RESTARTS   AGE   NODE
nginx-58cd5b85f7-4fz88     0/1     Evicted   0          1h    node1
nginx-58cd5b85f7-ts7ps     1/1     Running   0          12m   node2
```

You need to check the status by describing the node.

```
$ kubectl describe no node1
Taints:             node.kubernetes.io/disk-pressure:NoSchedule
Conditions:
  Type                    Status   LastHeartbeatTime                    LastTransitionTime
Reason                    Message
  ----                  ------  -----------------                ------------------              --
----                         -------
  DiskPressure           True     Tue, 20 Oct 2020 12:22:14 +0900    Wed, 14 Oct 2020 20:37:51
+0900    KubeletHasDiskPressure          kubelet has disk pressure
```

You should check if DistPressure is caused by insufficient disk space on the node and if so, free up the space.

```
$ df -h /var/lib/kubelet
Filesystem                    Size   Used Avail Use% Mounted on
/dev/mapper/VGROOT-LV_root     50G    46G   4.7G   91% /
```

Nodes are usually evicted when the disk to which the /var/lib/kubelet area belongs is 90% full or more (when using the default kubelet policy).

3.11 Restart frequently occurs

There are cases with frequent restarts even when the pod is running normally.

```
$ kubectl get po –n kube-system
NAME                  READY     STATUS       RESTARTS     AGE
```

```
$ kubectl logs weave-net-kwzvz -n kube-system –c weave --previous
FATA:    2020/10/15    04:25:03.297881    [kube-peers]    Could    not    get    peers:    Get
https://172.24.0.1:443/api/v1/nodes: dial tcp 172.24.0.1:443: i/o timeout
Failed to get peers
```

If the problem can be resolved by checking the previous pod logs with the --previous option, you should troubleshoot the application.

Common issues include communication failure with an external DB service or an internal pod related to the application causing restarts due to normal node work or network PM work. If the previous pod log cannot be confirmed, you need to check Last Stated: using the describe command.

```
$ kubectl describe po <pod-name>
    Last State:      Terminated
      Reason:          OOMKilled
      Exit Code:      137
```

In the case of a shutdown due to OOM (OutOfMemory), pod resources should be configured considering service availability.

```
$ kubectl describe po <pod-name>
Events:
  Type      Reason      Age                   From                  Message
  ----      ------      ----                  ----                  -------
  Warning   Unhealthy   48m  (x6  over  49m)    kubelet,  node2         Liveness  probe  failed:
localhost:5432 – no response
```

If the liveness probe fails, the liveness probe threshold should be set in consideration of the load situation and the connection lease.

3.12 If pod itself cannot be queried

This is when you create a workload such as Deployment but the pod cannot be queried.

In this case, you need to look up the Event object first.

```
$ kubectl get ev
LAST SEEN    TYPE       REASON              OBJECT                           MESSAGE
3m1s         Warning    FailedCreate        replicaset/nginx-847f85d779   Error creating: pods
"nginx-847f85d779-j484f"   is   forbidden:   exceeded   quota:   resourcequota,   requested:
requests.cpu=100m,requests.memory=2Gi, used: requests.cpu=4,requests.memory=128Mi, limited:
requests.cpu=2,requests.memory=1Gi
```

In the case above, ResourceQuota is set in the Namespace and the corresponding pod exceeds the quota or the LimitRange. You need to adjust either the ResourceQuota/LimitRange or pod resources.

```
$ kubectl get ev
LAST SEEN    TYPE       REASON              OBJECT                     MESSAGE
3s           Warning    FailedCreate        replicaset/nginx-56b5449445   Error creating: pods
"nginx-56b5449445-" is forbidden: error looking up service account default/my-sa: serviceaccount
"my-sa" not found
```

In this case, the resource required for running the pod has not been created. Necessary resources need to be created by looking up the pod specification.

```
$ kubectl get po –n kube-system
NAME                                READY    STATUS                RESTARTS    AGE
kube-controller-manager-master 0/1       CrashLoopBackOff    6              10m
```

If kube-controller-manager is abnormal, you need to check the log of kube-controller-manager and take necessary action.

4. Service status check

If there is no problem with the running state of the pod, you need to check the status of the service. First, you need to check if there is any issue in the communications to the pod.

4.1 Checking pod communication

Kubernetes nodes can directly send a communication request to <Pod IP>:<Port>.

```
$ kubectl get po -owide
NAME                          READY    STATUS      RESTARTS    AGE       IP
```

```
$ kubectl port-forward <pod-name> <local-port>:<pod-port>
Forwarding from 127.0.0.1:8888 -> 80
Forwarding from [::1]:8888 -> 80
$ curl localhost:8888
<title>Welcome to nginx!</title>
```

If the node is not a Kubernetes node, the communication status may be checked through port-forward of kubectl as shown below (only for TCP).

After pod communication is confirmed, you can check service communication.

4.2 Pod communication failure

However, if communication with the pod is not available, you need to first check if the pod is listening to the corresponding port.
In the case of normal port listening, the cause may be a communication block by network policies. You need to check communication from other pods that have accepted network policies.

```
$ kubectl get netpol
NAME              POD-SELECTOR    AGE
default-deny     <none>              1d
```

Alternatively, you could check if the Container Network Interface (CNI) Plugin is operating normally and take appropriate action.

```
$ kubectl get po –n kube-system | grep weave
NAME                     READY    STATUS                RESTARTS    AGE
weave-net-97gcr       1/2        CrashLoopBackOff    8              16m
```

4.3 Check service communication

As with pods, Kubernetes nodes also allow direct communication checks with <Service IP>:<Port>.

```
$ kubectl get svc
NAME             TYPE            CLUSTER-IP          EXTERNAL-IP     PORT(S)     AGE
nginx            ClusterIP     10.100.215.120     <none>              80/TCP       54m
$ curl 10.100.215.120:80
<title>Welcome to nginx!</title>
```

If they are not Kubernetes nodes, you can check communication status through kubectl port-forward (only for TCP).

```
$ kubectl port-forward service/<service-name> <local-port>:<pod-port>
Forwarding from 127.0.0.1:8888 -> 80
Forwarding from [::1]:8888 -> 80
$ curl localhost:8888
<title>Welcome to nginx!</title>
```

If service communication has been confirmed up to this point, you can proceed to Ingress communication check.

4.4 Service communication failure

If communication with the service fails, you can try the following inspection items.

```
$ kubectl describe svc <service-name>
Selector:              name=nginx
Port:                  http    80/TCP
TargetPort:            8080/TCP
Endpoints:              <none>
```

When describing a service, it is necessary to check whether or not endpoints have been used. If it is <none>, the service selector and the pod label may have been mismatched and they must be corrected.

Below displays a communication failure even with an endpoint.

```
$ kubectl describe svc <service-name>
Selector:            name=nginx
Port:                http   80/TCP
TargetPort:          8080/TCP
Endpoints:            10.36.0.1:8080
```

If the targetPort of the service and the containerPort of the pod are mismatched, the ports must be modified to match. If this is not the case, you should check if communication is disabled due to network policies, see if kube-proxy is abnormal, and then take following measures.

5. Ingress health check

If maintenance on the service has been completed, you can finish checking workloads related to Kubernetes after checking communication with Ingress.

5.1 Check Ingress communication

Under normal circumstances, pod IP information should be present in the backend during describing.

```
$ kubectl describe ing <ingress-name>
Name:                 nginx
Address:             172.28.128.11
Rules:
   Host                   Path   Backends
   ----                  ----   --------
   nginx.example.io
                              nginx:80 (10.36.0.1:80,10.44.0.2:80)
```

You can check the domain set in Ingress by directly accessing it through a web browser or with the curl command on the server. If the domain is not registered in the DNS server, you can check it by adding the domain to the header as shown below.

```
$ curl -H 'Host: <host-domain>' http://<ingress-controller-external-ip>
<title>Welcome to nginx!</title>
```

If communications have worked up to this point, there seems to be no issue within the Kubernetes cluster.

5.2 Ingress communication failure

There are also cases where communication to Ingress fails.

First, there could be no pod IP information in the Ingress backend.

```
$ kubectl describe ing <ingress-name>
Name:                nginx
Address:             172.28.128.11
Rules:
  Host                 Path   Backends
  ----                 ----   --------
  nginx.example.io
                          nginx:80 (<none>)
```

In this case, you should confirm if the serviceName and servicePort in the backend of the Ingress spec are properly matched with the actual service resource.

From here on are cases where communication with Ingress backend information is not working.

```
$ curl -H 'Host: nginx.example.io' http://172.28.128.11
<title>504 Gateway Time-out</title>
```

For the case below, you need to check if communication has been blocked by network policies.

```
$ curl -H 'Host: nginx.example.io' http://172.28.128.11
<title>503 Service Temporarily Unavailable</title>
```

If the application service is not responding, you need to check if the pod status is normal.

```
$ curl -H 'Host: nginx.example.io' http://172.28.128.11
<title>404 Not Found</title>
```

You must check if the path set in the Ingress backend is the context path provided by the actual application.

```
$ curl -H 'Host: nginx.example.io' http://172.28.128.11
<title>404 Not Found</title>
```

You need to make sure that the Ingress Controller and firewall are open.

```
$ curl -H 'Host: nginx.example.io' http://172.28.128.11
curl: (7) Failed connect to 172.28.128.11:80; Connection refused
```

This may occur when the Ingress Controller does not listen to the port. You should check if the Ingress Controller is running normally.

Additionally, if it is blocked by a firewall from the outside, check communication through the command below and then open <ingress-controller-external-ip>:80,443 and the firewall.

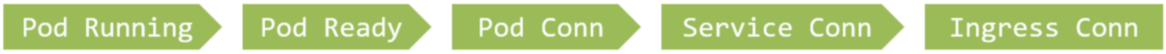
```
$ telnet <ingress-controller-external-ip> 80
or
$ </dev/tcp/<ingress-controller-external-ip>/80
```

If the domain is not registered externally, a “Could not resolve host” error may occur. After checking if the domain issue can be resolved using the following command, you need to register the domain in the DNS server or individual domains in the host files before use.

```
$ nslookup <host-domain>
or
$ getent hosts <host-domain>
```

6. Quick inspection guide

The following is the order of a quick inspection based on the guidelines so far.



1. Check Running and Ready states of pods

```
$ kubectl get po
NAME                       READY   STATUS    RESTARTS   AGE
nginx-58cd5b85f7-84xlr     1/1     Running   0          31s
```

2. Check pod communication

```
$ curl <Pod IP>:<port>
```

3. Check service communication

```
$ curl <Service IP>:<port>
```

4. Check Ingress communication

```
$ curl <Service IP>:<port>
```