Kubernetes 部署失败的 10 个最普遍原因（一）

在过去的两年间，我和许多团队合作，一起用 Kubernetes 来部署他们的应用。要想让开发者跟上 Kubernetes 术语的发展速度是很困难的，因此当部署失败的时候，我总是被要求指出哪个地方错了。

和客户一起工作，我的一个主要目标是自动化，尽可能把自己从繁琐的定位工作中解放出来。因此我努力给开发者必要的工具，使得他们自己就能定位部署失败的原因。我总结了 Kubernetes 部署失败的最普遍的原因，下面将和你分享我的定位过程。

言归正传，以下就是 Kubernetes 部署失败的十大原因。

1. 错误的容器镜像/非法的仓库权限

其中两个最普遍的问题是：(a)指定了错误的容器镜像，(b)使用私有镜像却不提供仓库认证信息。这在首次使用 Kubernetes 或者绑定 CI/CD 环境时尤其棘手。

让我们看个例子。首先我们创建一个名为 fail 的 deployment，它指向一个不存在的 Docker 镜像：

```
$ kubectl run fail --image=rosskukulinski/dne:v1.0.0
```

然后我们查看 Pods，可以看到有一个状态为 ErrImagePull 或者 ImagePullBackOff 的 Pod：

```
$ kubectl get pods
NAME                READY     STATUS             RESTARTS   AGE
fail-1036623984-hxoas   0/1       ImagePullBackOff   0          2m
```

想查看更多信息，可以 describe 这个失败的 Pod：

```
$ kubectl describe pod fail-1036623984-hxoas
```

查看 describe 命令的输出中 Events 这部分，我们可以看到如下内容：

```
Events:
FirstSeen    LastSeen    Count    From                    SubObjectPath       Type        Reason
 Message
---------    --------    -----    ----                    -------------       --------    ------     -------
```

5m 5m 1 {default-scheduler } Normal Scheduled Successfully assigned fail-1036623984-hxoas to gke-nrhk-1-default-pool-a101b974-wfp7
5m 2m 5 {kubelet gke-nrhk-1-default-pool-a101b974-wfp7} spec.containers{fail} Normal Pulling pulling image "rosskukulinski/dne:v1.0.0"
5m 2m 5 {kubelet gke-nrhk-1-default-pool-a101b974-wfp7} spec.containers{fail} Warning Failed Failed to pull image "rosskukulinski/dne:v1.0.0": Error: image rosskukulinski/dne not found
5m 2m 5 {kubelet gke-nrhk-1-default-pool-a101b974-wfp7} Warning FailedSync Error syncing pod, skipping: failed to "StartContainer" for "fail" with ErrImagePull: "Error: image rosskukulinski/dne not found"

5m 11s 19 {kubelet gke-nrhk-1-default-pool-a101b974-wfp7} spec.containers{fail} Normal BackOff Back-off pulling image "rosskukulinski/dne:v1.0.0"
5m 11s 19 {kubelet gke-nrhk-1-default-pool-a101b974-wfp7} Warning FailedSync Error syncing pod, skipping: failed to "StartContainer" for "fail" with ImagePullBackOff: "Back-off pulling image \"rosskukulinski/dne:v1.0.0\""

显示错误的那句话：Failed to pull image "rosskukulinski/dne:v1.0.0": Error: image rosskukulinski/dne not found 告诉我们 Kubernetes无法找到镜像 rosskukulinski/dne:v1.0.0。

因此问题变成：为什么 Kubernetes 拉不下来镜像?

除了网络连接问题外，还有三个主要元凶：

- 镜像 tag 不正确
- 镜像不存在（或者是在另一个仓库）
- Kubernetes 没有权限去拉那个镜像

如果你没有注意到你的镜像 tag 的拼写错误，那么最好就用你本地机器测试一下。

通常我会在本地开发机上，用 docker pull 命令，带上 完全相同的镜像 tag，来跑一下。比如上面的情况，我会运行命令 docker pull rosskukulinski/dne:v1.0.0。

- 如果这成功了，那么很可能 Kubernetes 没有权限去拉取这个镜像。参考镜像拉取 Secrets（http://suo.im/2gPTyG）来解决这个问题。

- 如果失败了，那么我会继续用不显式带 tag 的镜像测试 - docker pull rosskukulinski/dne - 这会尝试拉取 tag 为 latest 的镜像。如果这样成功，表明原来指定的 tag 不存在。这可能是人为原因，拼写错误，或者 CI/CD 的配置错误。

如果 docker pull rosskukulinski/dne（不指定 tag）也失败了，那么我们碰到了一个更大的问题：我们所有的镜像仓库中都没有这个镜像。默认情况下，Kubernetes 使用 Dockerhub 镜像仓库，如果你在使用 Quay.io，AWS ECR，或者 Google Container Registry，你要在镜像地址中指定这个仓库的 URL，比如使用 Quay，镜像地址就变成 quay.io/rosskukulinski/dne:v1.0.0。

如果你在使用 Dockerhub，那你应该再次确认你发布镜像到 Dockerhub 的系统，确保名字和 tag 匹配你的 deployment 正在使用的镜像。

注意：观察 Pod 状态的时候，镜像缺失和仓库权限不正确是没法区分的。其它情况下，Kubernetes 将报告一个 ErrImagePull 状态。

2. 应用启动之后又挂掉

无论你是在 Kubernetes 上启动新应用，还是迁移应用到已存在的平台，应用在启动之后就挂掉都是一个比较常见的现象。

我们创建一个 deployment，它的应用会在1秒后挂掉：

```
$ kubectl run crasher --image=rosskukulinski/crashing-app
```

我们看一下 Pods 的状态：

```
$ kubectl get pods
NAME                       READY     STATUS             RESTARTS   AGE
crasher-2443551393-vuehs   0/1       CrashLoopBackOff   2          54s
```

CrashLoopBackOff 告诉我们，Kubernetes 正在尽力启动这个 Pod，但是一个或多个容器已经挂了，或者正被删除。

让我们 describe 这个 Pod 去获取更多信息：

```
$ kubectl describe pod crasher-2443551393-vuehs
```

```
Name:         crasher-2443551393-vuehs
Namespace:    fail
Node:         gke-nrhk-1-default-pool-a101b974-wfp7/10.142.0.2
Start Time:   Fri, 10 Feb 2017 14:20:29 -0500
Labels:       pod-template-hash=2443551393
              run=crasher
Status:       Running
IP:           10.0.0.74
Controllers:  ReplicaSet/crasher-2443551393
Containers:
crasher:
Container ID:
docker://51c940ab32016e6d6b5ed28075357661fef3282cb3569117b0f815a199d01c60
Image:        rosskukulinski/crashing-app
Image ID:
docker://sha256:cf7452191b34d7797a07403d47a1ccf5254741d4bb356577b8a5de40864653a5
Port:
State:        Terminated
  Reason:     Error
  Exit Code:  1
  Started:    Fri, 10 Feb 2017 14:22:24 -0500
  Finished:   Fri, 10 Feb 2017 14:22:26 -0500
Last State:   Terminated
  Reason:     Error
  Exit Code:  1
  Started:    Fri, 10 Feb 2017 14:21:39 -0500
  Finished:   Fri, 10 Feb 2017 14:21:40 -0500
Ready:        False
Restart Count:  4
...
```

好可怕，Kubernetes 告诉我们这个 Pod 正被 Terminated，因为容器里的应用挂了。我们还可以看到应用的 Exit Code 是 1。后面我们可能还会看到一个 OOMKilled 错误。

我们的应用正在挂掉？为什么？

首先我们查看应用日志。假定你发送应用日志到 stdout（事实上你也应该这么做），你可以使用 kubectl logs 看到应用日志:

```
$ kubectl logs crasher-2443551393-vuehs
```

不幸的是，这个 Pod 没有任何日志。这可能是因为我们正在查看一个新起的应用实例，因此我们应该查看前一个容器：

```
$ kubectl logs crasher-2443551393-vuehs --previous
```

什么！我们的应用仍然不给我们任何东西。这个时候我们应该给应用加点启动日志了，以帮助我们定位这个问题。我们也可以本地运行一下这个容器，以确定是否缺失环境变量或者挂载卷。

3. 缺失 ConfigMap 或者 Secret

Kubernetes 最佳实践建议通过 ConfigMaps（https://kubernetes.io/docs/user-guide/configmap/）或者 Secrets 传递应用的运行时配置。这些数据可以包含数据库认证信息，API endpoints，或者其它配置信息。

一个常见的错误是，创建的 deployment 中引用的 ConfigMaps 或者 Secrets 的属性不存在，有时候甚至引用的 ConfigMaps 或者 Secrets 本身就不存在。

缺失 ConfigMap

第一个例子，我们将尝试创建一个 Pod，它加载 ConfigMap 数据作为环境变量：

```
# configmap-pod.yaml
apiVersion: v1
kind: Pod
metadata:
name: configmap-pod
spec:
containers:
- name: test-container
  image: gcr.io/google_containers/busybox
```

```
  command: [ "/bin/sh", "-c", "env" ]
  env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: special.how
```

让我们创建一个 Pod：kubectl create -f configmap-pod.yaml。在等待几分钟之后，我们可以查看我们的 Pod：

```
$ kubectl get pods
NAME            READY     STATUS              RESTARTS   AGE
configmap-pod   0/1       RunContainerError   0          3s
```

Pod 状态是 RunContainerError 。我们可以使用 kubectl describe 了解更多：

```
$ kubectl describe pod configmap-pod
[...]
Events:
FirstSeen    LastSeen    Count    From                        SubObjectPath          Type        Reason
 Message
---------    --------    -----    ----                -------------        --------    ------     -------
20s        20s     1    {default-scheduler }                          Normal      Scheduled    Successfully
assigned configmap-pod to gke-ctm-1-sysdig2-35e99c16-tgfm
19s            2s          3     {kubelet  gke-ctm-1-sysdig2-35e99c16-tgfm}     spec.containers{test-
container} Normal      Pulling      pulling image "gcr.io/google_containers/busybox"
18s            2s          3     {kubelet  gke-ctm-1-sysdig2-35e99c16-tgfm}     spec.containers{test-
container}     Normal                Pulled                Successfully    pulled    image
"gcr.io/google_containers/busybox"
18s          2s        3    {kubelet gke-ctm-1-sysdig2-35e99c16-tgfm}                  Warning
FailedSync   Error syncing pod, skipping: failed to "StartContainer" for "test-container" with
RunContainerError: "GenerateRunContainerOptions: configmaps \"special-config\" not found"
```

Events 章节的最后一条告诉我们什么地方错了。Pod 尝试访问名为 special-config 的 ConfigMap，但是在该 namespace 下找不到。一旦我们创建这个 ConfigMap，Pod 应该重启并能成功拉取运行时

数据。

在 Pod 规格说明中访问 Secrets 作为环境变量会产生相似的错误，就像我们在这里看到的ConfigMap错误一样。

但是假如你通过 Volume 来访问 Secrets 或者 ConfigMap会发生什么呢?

缺失 Secrets

下面是一个pod规格说明，它引用了名为 myothersecret 的 Secrets，并尝试把它挂为卷：

```
# missing-secret.yaml
apiVersion: v1
kind: Pod
metadata:
name: secret-pod
spec:
containers:
- name: test-container
  image: gcr.io/google_containers/busybox
  command: [ "/bin/sh", "-c", "env" ]
  volumeMounts:
    - mountPath: /etc/secret/
      name: myothersecret
restartPolicy: Never
volumes:
- name: myothersecret
  secret:
    secretName: myothersecret
```

让我们用 kubectl create -f missing-secret.yaml 来创建一个 Pod。

几分钟后，我们 get Pods，可以看到 Pod 仍处于 ContainerCreating 状态：

```
$ kubectl get pods
NAME          READY     STATUS              RESTARTS   AGE
```

```
secret-pod   0/1       ContainerCreating   0          4h
```

这就奇怪了。我们 describe 一下，看看到底发生了什么：

```
$ kubectl describe pod secret-pod
Name:        secret-pod
Namespace:    fail
Node:        gke-ctm-1-sysdig2-35e99c16-tgfm/10.128.0.2
Start Time:    Sat, 11 Feb 2017 14:07:13 -0500
Labels:
Status:        Pending
IP:
Controllers:

[...]

Events:
FirstSeen    LastSeen    Count    From                        SubObjectPath    Type        Reason
 Message
---------    --------    -----    ----                ------------    --------    ------     -------
18s        18s     1    {default-scheduler }                    Normal      Scheduled    Successfully
assigned secret-pod to gke-ctm-1-sysdig2-35e99c16-tgfm
18s        2s      6    {kubelet gke-ctm-1-sysdig2-35e99c16-tgfm}        Warning     FailedMount
MountVolume.SetUp failed for volume "kubernetes.io/secret/337281e7-f065-11e6-bd01-
42010af0012c-myothersecret" (spec.Name: "myothersecret") pod "337281e7-f065-11e6-
bd01-42010af0012c" (UID: "337281e7-f065-11e6-bd01-42010af0012c") with: secrets
"myothersecret" not found
```

Events 章节再次解释了问题的原因。它告诉我们 Kubelet 无法从名为 myothersecret 的 Secret 挂卷。为了解决这个问题，我们可以创建 myothersecret ，它包含必要的安全认证信息。一旦 myothersecret 创建完成，容器也将正确启动。

4. 活跃度/就绪状态探测失败

在 Kubernetes 中处理容器问题时，开发者需要学习的重要一课是，你的容器应用是 running 状态，不代表它在工作。

Kubernetes 提供了两个基本特性，称作活跃度探测和就绪状态探测（http://suo.im/3rkheK）。本质上来说，活跃度/就绪状态探测将定期地执行一个操作（例如发送一个 HTTP 请求，打开一个 tcp 连接，或者在你的容器内运行一个命令），以确认你的应用和你预想的一样在工作。

如果活跃度探测失败，Kubernetes 将杀掉你的容器并重新创建一个。如果就绪状态探测失败，这个 Pod 将不会作为一个服务的后端 endpoint，也就是说不会流量导到这个 Pod，直到它变成 Ready。

如果你试图部署变更你的活跃度/就绪状态探测失败的应用，滚动部署将一直悬挂，因为它将等待你的所有 Pod 都变成 Ready。

这个实际是怎样的情况？以下是一个 Pod 规格说明，它定义了活跃度/就绪状态探测方法，都是基于8080端口对 /healthy 路由进行健康检查：

```
apiVersion: v1
kind: Pod
metadata:
name: liveness-pod
spec:
containers:
- name: test-container
image: rosskukulinski/leaking-app
  livenessProbe:
    httpGet:
      path: /healthz
      port: 8080
    initialDelaySeconds: 3
    periodSeconds: 3
  readinessProbe:
    httpGet:
      path: /healthz
      port: 8080
    initialDelaySeconds: 3
    periodSeconds: 3
```

让我们创建这个 Pod：kubectl create -f liveness.yaml，过几分钟后查看发生了什么：

```
$ kubectl get pods
NAME           READY     STATUS    RESTARTS   AGE
liveness-pod   0/1       Running   4          2m
```

2分钟以后，我们发现 Pod 仍然没处于 Ready 状态，并且它已被重启了4次。让我们 describe 一下查看更多信息：

```
$ kubectl describe pod liveness-pod
Name:         liveness-pod
Namespace:    fail
Node:         gke-ctm-1-sysdig2-35e99c16-tgfm/10.128.0.2
Start Time:   Sat, 11 Feb 2017 14:32:36 -0500
Labels:
Status:       Running
IP:           10.108.88.40
Controllers:
Containers:
test-container:
Container ID:
 docker://8fa6f99e6fda6e56221683249bae322ed864d686965dc44acffda6f7cf186c7b
Image:        rosskukulinski/leaking-app
Image ID:
 docker://sha256:7bba8c34dad4ea155420f856cd8de37ba9026048bd81f3a25d222fd1d53da8b7
Port:
State:        Running
  Started:      Sat, 11 Feb 2017 14:40:34 -0500
Last State:     Terminated
  Reason:       Error
  Exit Code:    137
  Started:      Sat, 11 Feb 2017 14:37:10 -0500
  Finished:     Sat, 11 Feb 2017 14:37:45 -0500
[...]
```

```
Events:
FirstSeen LastSeen Count From SubObjectPath Type Reason Message
--------- -------- ----- ---- ------------- -------- ------ -------
8m 8m 1 {default-scheduler } Normal Scheduled Successfully assigned liveness-pod to gke-ctm-1-sysdig2-35e99c16-tgfm
8m 8m 1 {kubelet gke-ctm-1-sysdig2-35e99c16-tgfm} spec.containers{test-container} Normal Created Created container with docker id 0fb5f1a56ea0; Security: [seccomp=unconfined]
8m 8m 1 {kubelet gke-ctm-1-sysdig2-35e99c16-tgfm} spec.containers{test-container} Normal Started Started container with docker id 0fb5f1a56ea0
7m 7m 1 {kubelet gke-ctm-1-sysdig2-35e99c16-tgfm} spec.containers{test-container} Normal Created Created container with docker id 3f2392e9ead9; Security: [seccomp=unconfined]
7m 7m 1 {kubelet gke-ctm-1-sysdig2-35e99c16-tgfm} spec.containers{test-container} Normal Killing Killing container with docker id 0fb5f1a56ea0: pod "liveness-pod_fail(d75469d8-f090-11e6-bd01-42010af0012c)" container "test-container" is unhealthy, it will be killed and re-created.
8m 16s 10 {kubelet gke-ctm-1-sysdig2-35e99c16-tgfm} spec.containers{test-container} Warning Unhealthy Liveness probe failed: Get http://10.108.88.40:8080/healthz: dial tcp 10.108.88.40:8080: getsockopt: connection refused
8m 1s 85 {kubelet gke-ctm-1-sysdig2-35e99c16-tgfm} spec.containers{test-contai
```

Events 章节再次救了我们。我们可以看到活跃度探测和就绪状态探测都失败了。关键的一句话是 container "test-container" is unhealthy, it will be killed and re-created。这告诉我们 Kubernetes 正在杀这个容器，因为容器的活跃度探测失败了。

这里有三种可能性：

1. 你的探测不正确，健康检查的 URL 是否改变了？

2. 你的探测太敏感了， 你的应用是否要过一会才能启动或者响应？

3. 你的应用永远不会对探测做出正确响应，你的数据库是否配置错了

查看 Pod 日志是一个开始调测的好地方。一旦你解决了这个问题，新的 deployment 应该就能成功了。

5. 超出CPU/内存的限制

Kubernetes 赋予集群管理员限制 Pod 和容器的 CPU 或内存数量的（https://kubernetes.io/docs/admin/limitrange/）能力。作为应用开发者，你可能不清楚这个限制，导致 deployment 失败的时候一脸困惑。

我们试图部署一个未知 CPU/memory 请求限额的 deployment：

```
# gateway.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
name: gateway
spec:
template:
metadata:
  labels:
    app: gateway
spec:
  containers:
    - name: test-container
      image: nginx
      resources:
        requests:
          memory: 5Gi
```

你会看到我们设了 5Gi 的资源请求（https://kubernetes.io/docs/admin/limitrange/）。让我们创建这个 deployment：kubectl create -f gateway.yaml。

现在我们可以看到我们的 Pod：

```
$ kubectl get pods
```

```
No resources found.
```

为啥，让我们用 describe 来观察一下我们的 deployment：

```
$ kubectl describe deployment/gateway
Name:           gateway
Namespace:      fail
CreationTimestamp:  Sat, 11 Feb 2017 15:03:34 -0500
Labels:         app=gateway
Selector:       app=gateway
Replicas:       0 updated | 1 total | 0 available | 1 unavailable
StrategyType:       RollingUpdate
MinReadySeconds:    0
RollingUpdateStrategy:  0 max unavailable, 1 max surge
OldReplicaSets:
NewReplicaSet:      gateway-764140025 (0/1 replicas created)
Events:
FirstSeen   LastSeen    Count   From                SubObjectPath   Type        Reason
 Message
---------  --------  -----  ----          -------------  --------  ------      -------
4m      4m      1   {deployment-controller }        Normal      ScalingReplic
```

基于最后一行，我们的 deployment 创建了一个 ReplicaSet（gateway-764140025） 并把它扩展到 1。这个是用来管理 Pod 生命周期的实体。我们可以 describe 这个 ReplicaSet：

```
$ kubectl describe rs/gateway-764140025
Name:       gateway-764140025
Namespace:  fail
Image(s):   nginx
Selector:   app=gateway,pod-template-hash=764140025
Labels:     app=gateway
    pod-template-hash=764140025
Replicas:   0 current / 1 desired
Pods Status:    0 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
```

Events:

FirstSeen LastSeen Count From SubObjectPath Type Reason Message

--- --- --- --- --- --- --- ---

6m 28s 15 {replicaset-controller } Warning FailedCreate Error creating: pods "gateway-764140025-" is forbidden: [maximum memory usage per Pod is 100Mi, but request is 5368709120., maximum memory usage per Container is 100Mi, but request is 5Gi.]

哈知道了。集群管理员设置了每个 Pod 的最大内存使用量为 100Mi（好一个小气鬼！）。你可以运行 kubectl describe limitrange 来查看当前租户的限制。

你现在有3个选择：

1. 要求你的集群管理员提升限额

2. 减少 deployment 的请求或者限额设置

3. 直接编辑限额