116Kubernetes 系列（一零九）会话亲和性和 Kubernetes——谨慎行事！

有时，您可能会发现，当您第一次运行超过 1 个副本时，您的 Kubernetes 应用程序无法正常工作。

根据我的经验，这种情况最常见，因为应用程序是“Pod 有状态的”——例如，应用程序可能将会话状态存储在内存中，或者在后续请求期间将所需的数据写入本地存储。初始请求被发送到 Pod A，状态被初始化，一些后续请求被发送到 Pod B，而 Pod B 无权访问该状态，因此应用程序的行为不符合预期.....

通常，最初的反应是“好吧，我怎样才能使该会话的所有请求返回到处理初始请求的 Pod？” Kubernetes Service 通过 ClientIP 支持亲和性。如果您通过 Ingress 访问应用程序，大多数 Ingress 控制器将支持称为会话亲和性的功能（示例 1（https://github.com/kubernetes/ingress-nginx/blob/main/docs/user-guide/nginx-configuration/annotations.md#session-affinity）、示例 2（https://traefik.io/glossary/what-are-sticky-sessions/）、示例 3（https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.2/guide/ingress/annotations/#custom-attributes））， 通常使用 cookie。

但是，请不要认为这些选择是灵丹妙药！值得注意的是，基于容器的应用程序设计的原则之一是进程可处置性。容器应该“尽可能是短暂的，并准备好在任何时间点被另一个容器实例替换”。例如，当您的容器将会话状态存储在内存中时，它们实际上无法被另一个容器实例替换（至少不会影响部分活跃用户）。这一原则有一些非常好的理由。

让我们看一个例子

考虑以下简单的 java servlet：

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

@WebServlet("/helloworld")
public class HelloWorldServlet extends HttpServlet {

    public static final String LANG_PARAM_NAME = "lang";

    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {

        HttpSession session = request.getSession();
        session.setAttribute(LANG_PARAM_NAME,
                request.getParameter(LANG_PARAM_NAME);
    }

    protected void doGet(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {

        HttpSession session = request.getSession();

        PrintWriter writer = response.getWriter();
        writer.println("Language: " + session.getAttribute(LANG_PARAM_NAME));
    }
}
```

在第一个 (POST) 请求中，应用程序从请求中检索语言参数并将其值存储在会话状态中。与此会话关联的后续 (GET) 请求期望从会话状态中获得语言参数，但只有当所有请求都路由到同一个 Pod 时才会出现 — 默认情况下，如果副本 > 1 则不会出现这种情况。

Service 会话亲和性

一种选择是在 Service 上实现会话亲和性：

```yaml
kind: Service
apiVersion: v1
metadata:
  name: myservice
spec:
  selector:
    app: myapp
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 80
  # The following adds session affinity
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 600
```

这在直接访问 ClusterIP Service 以及使用 LoadBalancer 类型服务时有效。不幸的是，许多高可用的 NodePort 服务和 Ingress 配置在默认情况下可能无法工作，因为 Service 获取的 ClientIP 将是负载均衡器或 Ingress 控制器 Pod 的 IP，而不是客户端的 IP。

Ingress 会话亲和性

如果您的请求是通过 Ingress 发出的，另一种选择是使用 Ingress 提供的 cookie 会话亲和性。有许多不同的 Ingress 实现，但我们只看一个 - 用于 Kubernetes 的 Ingress-NGINX 控制器 - 作为示例：

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: helloworld-deployment-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/affinity: "cookie"
    nginx.ingress.kubernetes.io/session-cookie-path: "/"
spec:
  ingressClassName: myingressclass
  rules:
  - http:
      paths:
      - path: /helloworld/(.*)
        pathType: Prefix
        backend:
          service:
            name: helloworld-deployment-svc
            port:
              number: 8080
```

问题解决了.....对吧？？？

实施上述“解决方案”之一后，您尝试点击应用程序，然后.....成功！正确的...？

请不要那么快！

当然，这使事情变得更好.....但是，在很多情况下，内存中的会话状态将不可用，并且这些情况通常不是您正在显式测试的情况。

Worker 节点重启

您的工作节点有时会重新启动，有时这将超出您的控制范围。可能会发生计划内的修补、升级和其他维护，但硬件故障（即使是主要的云提供商）可能而且确实会在没有警告的情况下发生。不幸的是，当工作节点重新启动时，该工作节点上运行的任何 Pod 中存在的状态都将消失，用户将受到影响。

容器重启

如果容器的 livenessProbe 失败，它将重新启动。应用程序可能会尝试消耗超过其配置限制的内存，并且容器可能会被 OOMKilled。容器进程可能会崩溃。不幸的是，当容器重新启动时，状态消失，用户将受到影响。

自动缩放

HorizontalPodAutoscaling 自动更新工作负载资源（例如 Deployments 或 StatefulSet），目的是扩展资源以满足需求。这意味着当应用程序繁忙时添加新的 Pod，并在应用程序不太繁忙时删除 Pod。不幸的是，当 HPA 删除 Pod 时，用户将会受到

影响。

VerticalPodAutoscaling 还会导致容器在可能意外的时间重新启动。它根据使用情况自动更新容器请求，以允许容器访问它们所需的资源，并允许将 Pod 正确调度到节点上，以便提供适当的资源量。然而，每次更新请求时，容器都会重新启动，用户都会受到影响。

工作节点资源缺乏

不幸的是，计算资源不是无限的——有时工作节点可能会发现自己没有足够的资源来调度 Pod。这可能会导致节点压力驱逐，即 Kubernetes 主动终止 Pod 以回收节点上的资源，并根据可用容量以更优化的方式理想地分配工作负载。不幸的是，当 Pod 被终止并重新调度到其他工作节点上时，用户将受到影响。

应用程序变更的推出

当应用程序发生更改 Pod 规范时，Pod 将被终止，并安排一个新的 Pod，以便更改生效（或者，有时，将先安排一个新的 Pod，然后终止现有的 Pod。这样，一个 Pod 就会消失，然后由一个替代 Pod 取代）。不幸的是，用户将再次受到影响。

更好的解决方案

我们通常应该尽量避免影响（不利的）应用程序用户。

考虑为会话状态使用外部缓存（示例 1（https://www.baeldung.com/spring-session）、示例 2（https://openliberty.io/docs/latest/reference/feature/sessionCache-1.0.html）、示例 3（https://learn.microsoft.com/en-us/aspnet/core/performance/caching/distributed?view=aspnetcore-7.0#distributed-redis-cache））或数据库（示例 1（https://www.ibm.com/docs/en/was-liberty/base?topic=manually-configuring-liberty-session-persistence-database）、示例 2（https://learn.microsoft.com/en-us/aspnet/core/performance/caching/distributed?view=aspnetcore-7.0#distributed-sql-server-cache）、示例 3（https://tomcat.apache.org/tomcat-9.0-doc/config/manager.html））。如果状态是文件，请考虑使用 NFS 或 CIFS/SMB 等共享文件系统，甚至可能使用 Kubernetes API 来更新 ConfigMap 或 Secret。如果供应商要求您为容器化应用程序配置会话亲和性，您可能会建议改进他们的应用程序！

无论哪种方式，请抵制仅仅因为可以而简单地配置会话亲和力的冲动——也许您应该删除会话亲和力的“要求”。