编辑dockerfile文件

FROM centos

#nginx

ADD nginx-1.6.0.tar.gz /usr/src/

RUN yum -y install gcc-c++ pcre pcre-devel zlib zlib-devel openssl openssl-devel vim net-tools make psmisc \

&& useradd nginx -M -s /sbin/nologin

WORKDIR /usr/src/nginx-1.6.0/

RUN ./configure --prefix=/usr/local/nginx --group=nginx --user=nginx && make && make -j4 install && ln -s /usr/local/nginx/sbin/\* /usr/local/sbin/

EXPOSE 80

EXPOSE 443

#CMD ["nginx","-g","daemon off;"]

#mysql

ADD cmake-2.8.6.tar.gz /usr/src

ADD mysql-5.6.36.tar.gz /usr/src

WORKDIR /usr/src/cmake-2.8.6

RUN ./configure && gmake && gmake install

RUN yum -y install ncurses-devel perl-Data-Dumper

RUN useradd -M -s /sbin/nologin mysql

WORKDIR /usr/src/mysql-5.6.36

RUN cmake -DCMAKE\_INSTALL\_PREFIX=/usr/local/mysql -DSYSCONFDIR=/etc -DDEFAULT\_CHARSET=utf8 -DDEFAULT\_COLLATION=utf8\_general\_ci -DWITH\_EXTRA\_CHARSETS=all && make -j4 && make install

RUN ln -s /usr/local/mysql/bin/\* /usr/local/bin/

RUN chown -R mysql:mysql /usr/local/mysql/

RUN /usr/local/mysql/scripts/mysql\_install\_db --user=mysql --group=mysql --basedir=/usr/local/mysql --datadir=/usr/local/mysql/data

RUN cp -p /usr/src/mysql-5.6.36/support-files/my-default.cnf /etc/my.cnf

RUN cp -p /usr/src/mysql-5.6.36/support-files/mysql.server /etc/rc.d/init.d/mysqld

RUN chmod a+x /etc/rc.d/init.d/mysqld

RUN chkconfig --add mysqld

RUN chkconfig mysqld on

EXPOSE 3306

#CMD ["/etc/init/mysqld start"]

#php

RUN yum -y install net-tools libxml2-devel zlib-devel openssl-devel libjpeg-devel libpng-devel gd

ADD php-5.5.38.tar.gz /usr/src

WORKDIR /usr/src/php-5.5.38

RUN ./configure --prefix=/usr/local/php --with-config-file-path=/usr/local/php --with-mysql=/usr/local/mysql --with-mysqli --enable-fpm --with-gd --enable-mbstring --with-jpeg-dir=/usr/lib --with-png-dir=usr/lib && make -j4 && make install

RUN ln -s /usr/local/php/sbin/\* /usr/local/sbin/

RUN ln -s /usr/local/php/bin/\* /usr/local/bin/

RUN cp -p /usr/src/php-5.5.38/php.ini-development /usr/local/php/php.ini

WORKDIR /usr/local/php/etc/

RUN cp -p php-fpm.conf.default php-fpm.conf

WORKDIR /usr/src/php-5.5.38/sapi/fpm

RUN cp -p init.d.php-fpm /etc/init.d/php-fpm

RUN chmod a+x /etc/init.d/php-fpm

#修改php-fpm配置文件

RUN sed -i 's/^;pid =.\*/pid = run\/php-fpm.pid/' /usr/local/php/etc/php-fpm.conf

RUN sed -i 's/^user =.\*/user = nginx/' /usr/local/php/etc/php-fpm.conf

RUN sed -i 's/^group =.\*/group = nginx/' /usr/local/php/etc/php-fpm.conf

RUN sed -i 's/^listen =.\*/listen = 0.0.0.0:9000/' /usr/local/php/etc/php-fpm.conf

EXPOSE 9000

#CMD ["/etc/init/php-fpm start"]

编辑nginx配置文件

location / {

root html;

index index.php index.html index.htm;

}

location ~ \.php$ {

root html;

fastcgi\_pass 172.17.0.2:9000;

fastcgi\_index index.php;

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name; include fastcgi\_params;

}

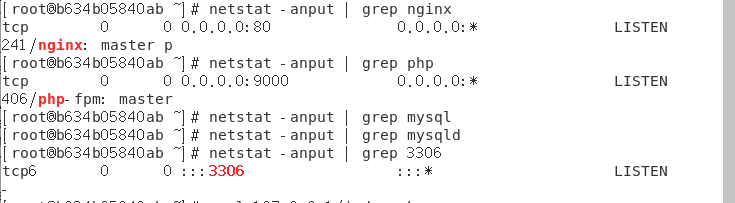
注意：：$document\_root$fastcgi\_script\_name;（自动找到其配置文件）

启动服务

nginx

/etc/init.d/mysqld restart

/etc/init.d/php-fpm restart



cat << EOF > /usr/local/nginx/html/index.php

<?php

\$link = mysqli\_connect('localhost','root','123.com');

if(\$link) echo "恭喜你，数据库连接成功啦！！";

?>

EOF



一般情况下我们部署的 Pod 是通过集群的自动调度策略来选择节点的，默认情况下调度器考虑的是资源足够，并且负载尽量平均，但是有的时候我们需要能够更加细粒度的去控制 Pod 的调度，比如我们内部的一些服务 gitlab 之类的也是跑在Kubernetes集群上的，我们就不希望对外的一些服务和内部的服务跑在同一个节点上了，担心内部服务对外部的服务产生影响；但是有的时候我们的服务之间交流比较频繁，又希望能够将这两个服务的 Pod 调度到同一个的节点上。这就需要用到 Kubernetes 里面的一个概念：亲和性和反亲和性。

亲和性有分成节点亲和性(nodeAffinity)和 Pod 亲和性(podAffinity)。

nodeSelector

在了解亲和性之前，我们先来了解一个非常常用的调度方式：nodeSelector。我们知道label是kubernetes中一个非常重要的概念，用户可以非常灵活的利用 label 来管理集群中的资源，比如最常见的一个就是 service 通过匹配 label 去匹配 Pod 资源，而 Pod 的调度也可以根据节点的 label 来进行调度。

不过需要注意的是nodeSelector属于强制性的，如果我们的目标节点没有可用的资源，我们的 Pod 就会一直处于 Pending 状态，这就是nodeSelector的用法。

亲和性和反亲和性调度

之前了解了 kubernetes 调度器的一个调度流程，我们知道默认的调度器在使用的时候，经过了 predicates 和 priorities 两个阶段，但是在实际的生产环境中，往往我们需要根据自己的一些实际需求来控制 pod 的调度，这就需要用到 nodeAffinity(节点亲和性)、podAffinity(pod 亲和性) 以及 podAntiAffinity(pod 反亲和性)。

亲和性调度可以分成软策略和硬策略两种方式:

软策略就是如果你没有满足调度要求的节点的话，pod 就会忽略这条规则，继续完成调度过程，说白了就是满足条件最好了，没有的话也无所谓了的策略

硬策略就比较强硬了，如果没有满足条件的节点的话，就不断重试直到满足条件为止，简单说就是你必须满足我的要求，不然我就不干的策略。

对于亲和性和反亲和性都有这两种规则可以设置： preferredDuringSchedulingIgnoredDuringExecution和requiredDuringSchedulingIgnoredDuringExecution，前面的就是软策略，后面的就是硬策略。

nodeAffinity

节点亲和性主要是用来控制 pod 要部署在哪些主机上，以及不能部署在哪些主机上的。它可以进行一些简单的逻辑组合了，不只是简单的相等匹配。

现在Kubernetes提供的操作符有下面的几种：

In：label 的值在某个列表中

NotIn：label 的值不在某个列表中

Gt：label 的值大于某个值

Lt：label 的值小于某个值

Exists：某个 label 存在

DoesNotExist：某个 label 不存在

如果nodeSelectorTerms下面有多个选项的话，满足任何一个条件就可以了；如果matchExpressions有多个选项的话，则必须同时满足这些条件才能正常调度 POD。

podAffinity

pod 亲和性主要解决 pod 可以和哪些 pod 部署在同一个拓扑域中的问题（其中拓扑域用主机标签实现，可以是单个主机，也可以是多个主机组成的 cluster、zone 等等），而 pod 反亲和性主要是解决 pod 不能和哪些 pod 部署在同一个拓扑域中的问题，它们都是处理的 pod 与 pod 之间的关系，比如一个 pod 在一个节点上了，那么我这个也得在这个节点，或者你这个 pod 在节点上了，那么我就不想和你待在同一个节点上。

这就是 pod 亲和性的用法，

而 pod 反亲和性则是反着来的，比如一个节点上运行了某个 pod，那么我们的 pod 则希望被调度到其他节点上去，同样我们把上面的 podAffinity 直接改成 podAntiAffinity，(pod-antiaffinity-demo.yaml)