学习笔记摘取版

2.1-2.6内容文字和解释性词语相对较多 在2.7中有总结和实际运用

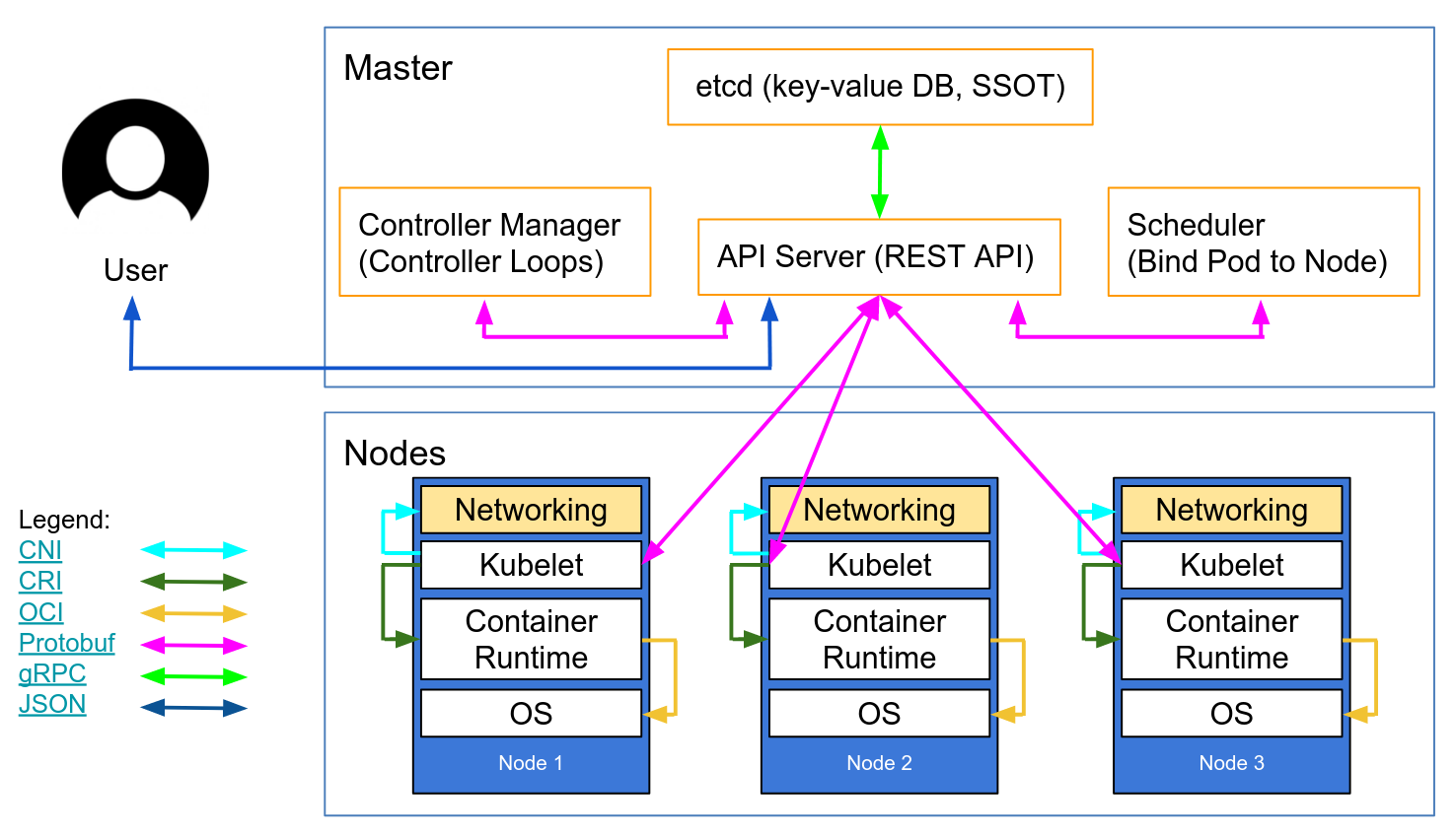
2.7 实战演练：玩转Kubernetes（1）

2.7.1基础知识回顾

接下来简要列举一下“初级篇”里讲到的 Kubernetes 要点，然后再综合运用这些知识，演示一个实战项目——还是搭建 WordPress 网站，不过这次不是在 Docker 里，而是在 Kubernetes 集群里。

Kubernetes 技术要点回顾容器技术开启了云原生的大潮，但成熟的容器技术，到生产环境的应用部署的时候，却显得“步履维艰”。因为容器只是针对单个进程的隔离和封装，而实际的应用场景却是要求许多的应用进程互相协同工作，其中的各种关系和需求非常复杂，在容器这个技术层次很难掌控。为了解决这个问题，容器编排（Container Orchestration）就出现了，它可以说是以前的运维工作在云原生世界的落地实践，本质上还是在集群里调度管理应用程序，只不过管理的主体由人变成了计算机，管理的目标由原生进程变成了**容器和镜像**。

而现在，容器编排领域的王者就是——Kubernetes。Kubernetes 源自 Borg 系统，它凝聚了 Google 的内部经验和 CNCF 的社区智慧，所以战胜了竞争对手 Apache Mesos 和 Docker Swarm，成为了容器编排领域的事实标准，也成为了云原生时代的基础操作系统，学习云原生就必须要掌握 Kubernetes。（10 讲）Kubernetes 的 **Master/Node** 架构是它具有自动化运维能力的关键，也对我们的学习至关重要，这里我再用另一张参考架构图来简略说明一下它的运行机制（图片来源）：



Kubernetes 把集群里的**计算资源定义为节点**（Node），其中又划分成控制面和数据面两类。

控制面是 **Master** 节点，负责管理集群和运维监控应用，里面的核心组件是 apiserver、etcd、scheduler、controller-manager。

数据面是 **Worker** 节点，受 Master 节点的管控，里面的核心组件是 kubelet、kube-proxy、container-runtime。

此外，Kubernetes 还支持**插件**机制，能够灵活扩展各项功能，常用的插件有 DNS 和 Dashboard。

为了更好地管理集群和业务应用，Kubernetes 从现实世界中抽象出了许多概念，称为“**API 对象**”，描述这些对象就需要使用 **YAML 语言**。YAML 是 JSON 的超集，但语法更简洁，表现能力更强，更重要的是它以“声明式”来表述对象的状态，不涉及具体的操作细节，这样 Kubernetes 就能够依靠存储在 etcd 里集群的状态信息，不断地“调控”对象，直至实际状态与期望状态相同，这个过程就是 Kubernetes 的自动化运维管理（11 讲）。

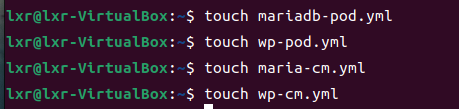
Kubernetes 里有很多的 API 对象，其中最核心的对象是“**Pod**”，它捆绑了一组存在密切协作关系的容器，容器之间共享网络和存储，在集群里必须一起调度一起运行。通过 Pod 这个概念，Kubernetes 就简化了对容器的管理工作，其他的所有任务都是通过对 Pod 这个最小单位的再包装来实现的（12 讲）。

除了核心的 Pod 对象，基于“单一职责”和“对象组合”这两个基本原则，我们又学习了 4 个比较简单的 API 对象，分别是 **Job/CronJob 和 ConfigMap/Secret**。Job/CronJob 对应的是离线作业，它们逐层包装了 Pod，添加了作业控制和定时规则（13 讲）。

ConfigMap/Secret 对应的是配置信息，需要以环境变量或者存储卷的形式注入进 Pod，然后进程才能在运行时使用（14 讲）。

和 Docker 类似，Kubernetes 也提供一个客户端工具，名字叫“**kubectl**”，它直接与 Master 节点的 **apiserver** 通信，把 **YAML** 文件发送给 **RESTful 接口**，从而触发 Kubernetes 的对象管理工作流程。kubectl 的命令很多，查看自带文档可以用 **api-resources、explain** ，查看对象状态可以用 **get、describe、logs** ，操作对象可以用 **run、apply、exec、delete** 等等（09 讲）。

使用 YAML 描述 API 对象也有固定的格式，必须写的“头字段”是**“apiVersion”“kind”“metadata”**，它们表示对象的**版本、种类和名字**等元信息。实体对象如 **Pod、Job、CronJob** 会再有“**spec**”字段描述对象的期望状态，最基本的就是容器信息，非实体对象如 **ConfigMap、Secret** 使用的是“**data**”字段，记录一些静态的字符串信息。好了，“初级篇”里的 Kubernetes 知识要点我们就基本总结完了，如果你发现哪部分不太清楚，可以课后再多复习一下前面的课程加以巩固。



2.7.2 k8s搭建wordpress

WordPress 网站搭建步骤了解基本架构之后，接下来我们就逐步搭建这个网站系统，总共需要 4 步。第一步当然是要**编排 MariaDB 对象**，它的具体运行需求可以参考“入门篇”的实战演练课，这里我就不再重复了。MariaDB 需要 4 个环境变量，比如数据库名、用户名、密码等，在 Docker 里我们是在命令行里使用参数 **--env**，而在 Kubernetes 里我们就应该使用 **ConfigMap**，为此需要定义一个 maria-cm 对象：

apiVersion: v1

kind: ConfigMap

metadata:

name: maria-cm

data:

DATABASE: 'db'

USER: 'wp'

PASSWORD: '123'

ROOT\_PASSWORD: '123'

然后我们**定义 Pod 对象 maria-pod**，把配置信息注入 Pod，让 MariaDB 运行时从环境变量读取这些信息：

apiVersion: v1

kind: Pod

metadata:

name: maria-pod

labels:

app: wordpress

role: database

spec:

containers:

- image: mariadb:10

name: maria

imagePullPolicy: IfNotPresent

ports:

- containerPort: 3306

envFrom:

- prefix: 'MARIADB\_'

configMapRef:

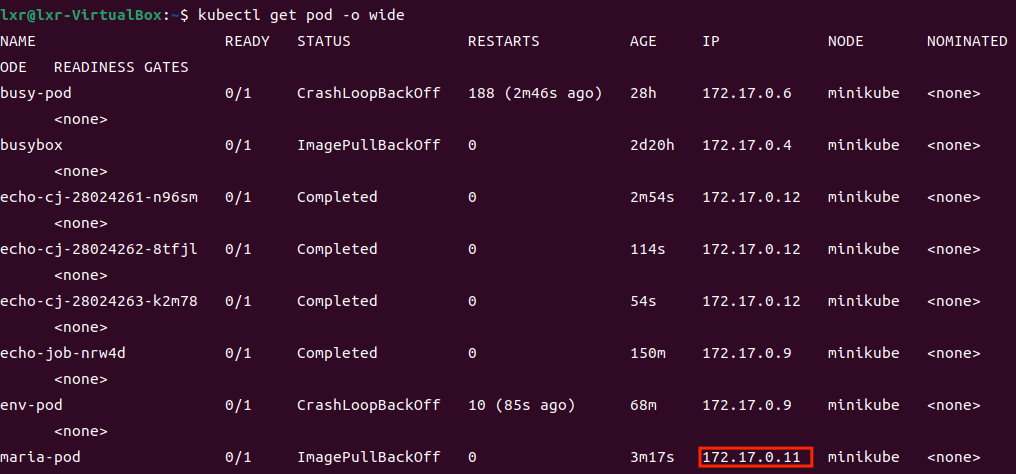
name: maria-cm

注意这里我们使用了一个新的字段“envFrom”，这是因为 ConfigMap 里的信息比较多，如果用 env.valueFrom 一个个地写会非常麻烦，容易出错，而 **envFrom** 可以一次性地把 ConfigMap 里的字段全导入进 Pod，并且能够指定变量名的前缀（即这里的 MARIADB\_），非常方便。

使用 kubectl apply 创建这个对象之后，可以用 kubectl get pod 查看它的状态，如果想要获取 IP 地址需要加上参数 -o wide ：

kubectl apply -f mariadb-pod.yml

kubectl get pod **-o wide**



现在数据库就成功地在 Kubernetes 集群里跑起来了，IP 地址是“172.17.0.11”，注意这个地址和 Docker 的不同，是 Kubernetes 里的私有网段。接着是第二步，**编排 WordPress** 对象，还是先用 ConfigMap 定义它的环境变量：

apiVersion: v1

kind: ConfigMap

metadata:

name: wp-cm

data:

HOST: '172.17.0.2'

USER: 'wp'

PASSWORD: '123'

NAME: 'db'

在这个 ConfigMap 里要注意的是“HOST”字段，它必须是 MariaDB Pod 的 IP 地址，如果不写正确 WordPress 会无法正常连接数据库。然后我们再编写 WordPress 的 YAML 文件，为了简化环境变量的设置同样使用了 envFrom：

apiVersion: v1

kind: Pod

metadata:

name: wp-pod

labels:

app: wordpress

role: website

spec:

containers:

- image: wordpress:5

name: wp-pod

imagePullPolicy: IfNotPresent

ports:

- containerPort: 80

envFrom:

- prefix: 'WORDPRESS\_DB\_'

configMapRef:

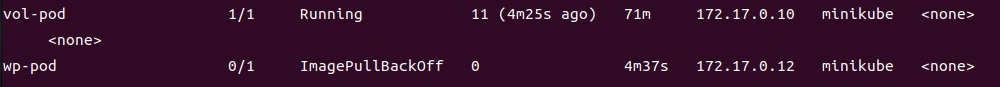
name: wp-cm

接着还是用 kubectl apply 创建对象，kubectl get pod 查看它的状态：

#查看POD IP地址 **备注pod-cm可以合并在一个yml文件，因为名字已经定义好了，但是另外定义的话需要执行其yml文件（kubectl apply -f xxx）**

kubectl apply -f wp-pod.yml

kubectl get pod **-o wide**

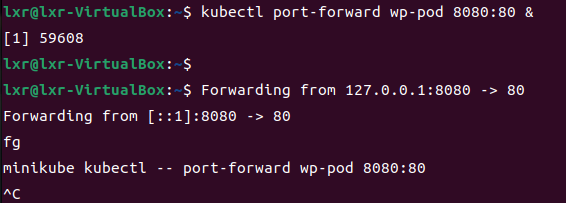
 第三步是**为 WordPress Pod 映射端口号**，让它在集群外可见。因为 Pod 都是运行在 Kubernetes 内部的私有网段里的，外界无法直接访问，想要对外暴露服务，需要使用一个专门的 kubectl port-forward 命令，它专门负责**把本机的端口映射到在目标对象的端口号，有点类似 Docker 的参数 -p**，经常用于 Kubernetes 的临时调试和测试。下面我就把本地的“8080”映射到 WordPress Pod 的“80”，kubectl 会把这个端口的所有数据都转发给集群内部的 Pod：

kubectl **port-forward** wp-pod 8080:80 **&**



注意在命令的末尾我使用了一个 & 符号，让端口转发工作**在后台进行**，这样就不会阻碍我们后续的操作。

如果想关闭端口转发，需要敲命令 fg ，它会把后台的任务带回到前台，然后就可以简单地用“Ctrl + C”来停止转发了。



第四步**是创建反向代理的 Nginx**，让我们的网站对外提供服务。这是因为 WordPress 网站使用了 **URL 重定向**，**直接使用“8080”会导致跳转故障**，所以为了让网站正常工作，我们还应该在 Kubernetes 之外启动 Nginx 反向代理，保证外界看到的仍然是**“80”端口号**。Nginx 的配置文件和第 7 讲基本一样，只是目标地址变成了“127.0.0.1:8080”，它就是我们在第三步里用 kubectl port-forward 命令创建的本地地址：

server {

listen 80;

default\_type text/html;

location / {

proxy\_http\_version 1.1;

proxy\_set\_header Host $host;

proxy\_pass http://127.0.0.1:8080;

}

}

然后我们用 docker run -v 命令加载这个配置文件，以容器的方式启动这个 Nginx 代理：

docker run -d --rm \

--net=host \

-v /tmp/proxy.conf:/etc/nginx/conf.d/default.conf \

nginx:alpine

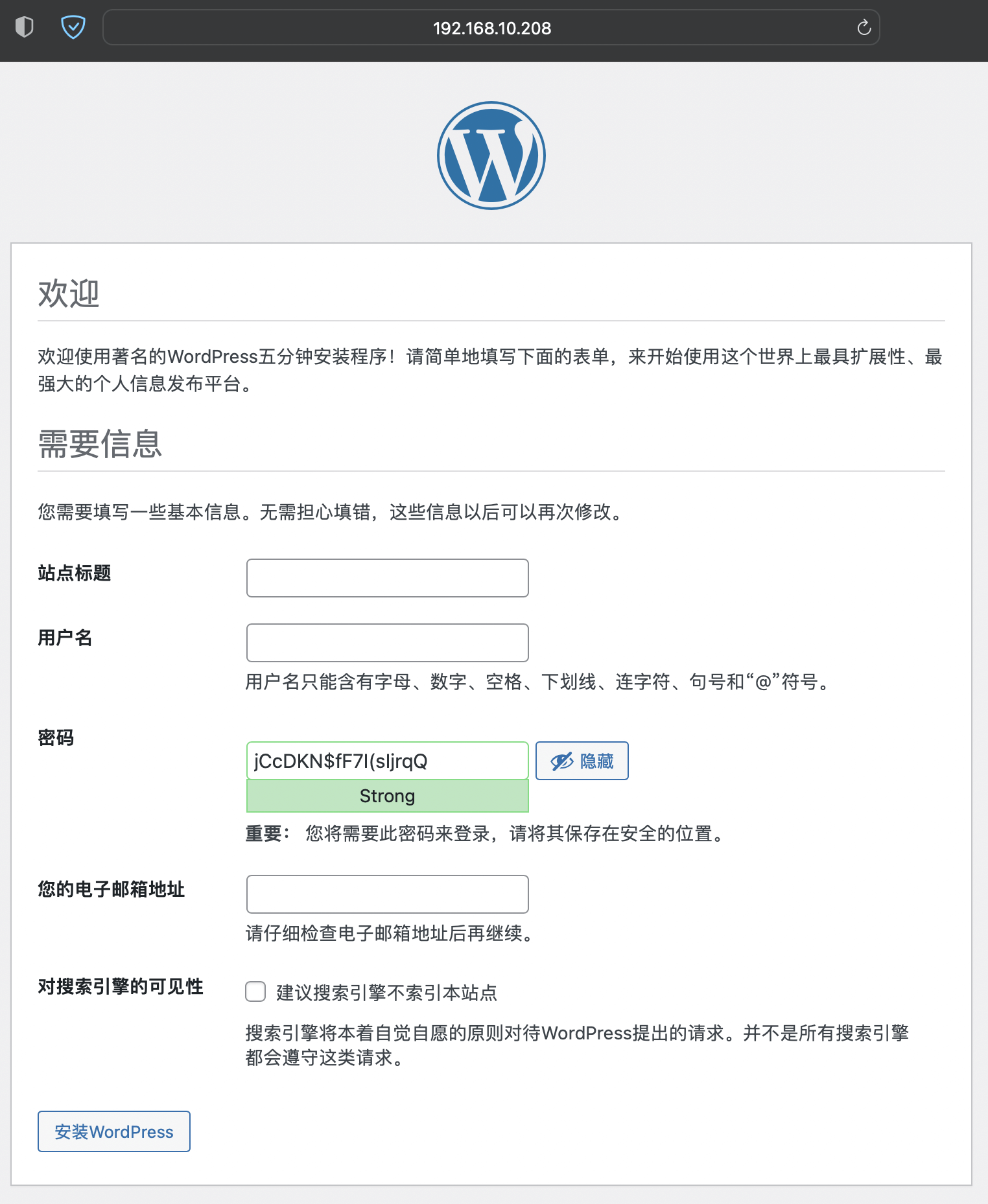
写一个脚本文件



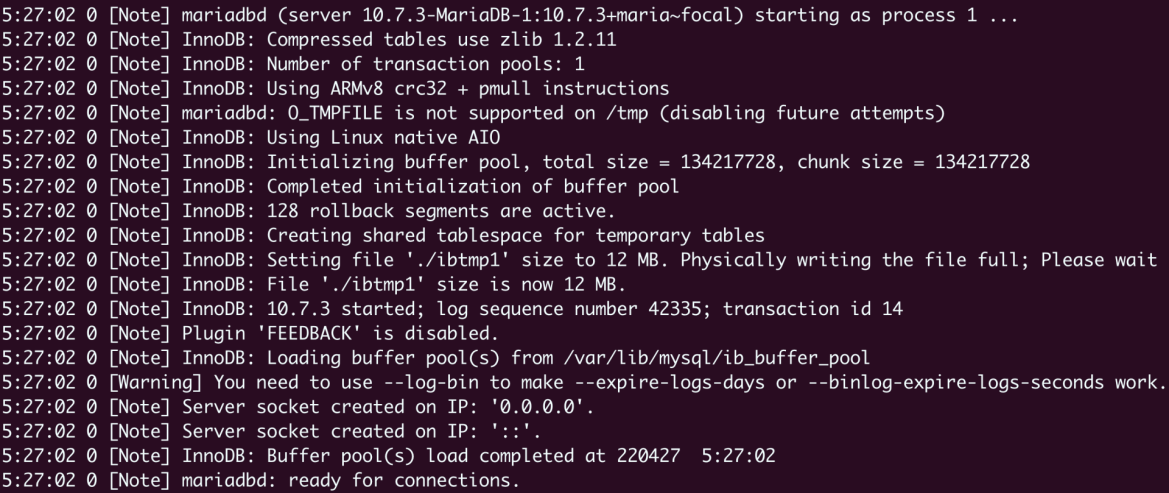
**\*其中相对原文档去掉了/tmp？**



有了 Nginx 的反向代理之后，我们就**可以打开浏览器**，输入本机的“127.0.0.1”或者是虚拟机的 IP 地址（我这里仍然是“http://192.168.10.208”），看到 WordPress 的界面：



也可以在 Kubernetes 里使用命令 kubectl logs 查看 WordPress、MariaDB 等 Pod 的运行日志，来验证它们是否已经正确地响应了请求：



使用 Dashboard 管理 Kubernetes

到这里 WordPress 网站就搭建成功了，我们的主要任务也算是完成了，不过我还想再带你看看 Kubernetes 的图形管理界面，也就是 Dashboard，看看不用命令行该怎么管理 Kubernetes。启动 Dashboard 的命令你还记得吗，在第 10 节课里讲插件的时候曾经说过，需要用 minikube，命令是：

minikube dashboard

它会自动打开浏览器界面，显示出当前 Kubernetes 集群里的工作负载：

