你好，我是秦粤。上节课我们只是用Docker部署了index.js，如果我们将所有拆解出来的微服务都用Docker独立部署，我们就要同时管理多个Docker容器，也就是Docker集群。如果是更复杂一些的业务，可能需要同时管理几十甚至上百个微服务，显然我们手动维护Docker集群的效率就太低了。而容器即服务CaaS，恰好也需要集群的管理工具。我们也知道FaaS的底层就是CaaS，那CaaS又是如何管理这么多函数实例的呢？怎么做才能提升效率？

我想你应该听过Kubernetes，它也叫K8s（后面统一简称K8s），用于自动部署、扩展和管理容器化应用程序的开源系统，是Docker集群的管理工具。为了解决上述问题，其实我们就可以考虑使用它。K8s的好处就在于，它具备跨环境统一部署的能力。

这节课，我们就试着在本地环境中搭建K8s来管理我们的Docker集群。但正常情况下，这个场景需要几台机器才能完成，而通过Docker，我们还是可以用一台机器就可以在本地搭建一个低配版的K8s。

下节课，我们还会在今天内容的基础上，用K8s的CaaS方式实现一套Serverless环境。通过这两节课的内容，你就可以完整地搭建出属于自己的Serverless了。

话不多说，我们现在就开始，希望你能跟着我一起多动手。

PC上的K8s

那在开始之前，我们先得把安装问题解决了，这部分可能会有点小困难，所以我也给你详细讲下。

首先我们需要安装kubectl，它是K8s的命令行工具。

你需要在你的PC上安装K8s，如果你的操作系统是MacOS或者Windows，那么就比较简单了，桌面版的Docker已经自带了K8s；其它操作系统的同学需要安装minikube。

不过，要顺利启动桌面版Docker自带的K8s，你还得解决国内Docker镜像下载不了的问题，这里请你先下载第8课的代码。接着，请你跟着我的步骤进行操作：

1. 开通阿里云的容器镜像仓库；
2. 在阿里云的容器镜像服务里，找到镜像加速器，复制你的镜像加速器地址；
3. 打开桌面版Docker的控制台，找到Docker Engine。

```
{
  "registry-mirrors" : [
    "https://你的加速地址.mirror.aliyuncs.com"
  ],
  "debug" : true,
  "experimental" : true
}
```

4. 预下载K8s所需要的所有镜像，执行我目录下的docker-k8s-prefetch.sh，如果你是Windows操作系统，建议使用gitBash[1]；

```
chmode +x docker-k8s-prefetch.sh
./docker-k8s-prefetch.sh
```

5. 上面拉取完运行K8s所需的Docker镜像，你就可以在桌面版Docker的K8s项中，勾选启动K8s了。

现在，K8s就能顺利启动了，启动成功后，请你继续执行下面的命令。

查看安装好的K8s系统运行状况。

```
kubectl get all -n kube-system
```

查看K8s有哪些配置环境，对应~/.kube/config。

```
kubectl config get-contexts
```

查看当前K8s环境的整体情况。

```
kubectl get all
```

按照我的流程走，在大部分的机器上本地运行K8s都是没有问题的，如果你卡住了，请在留言区告知我，我帮你解决问题。

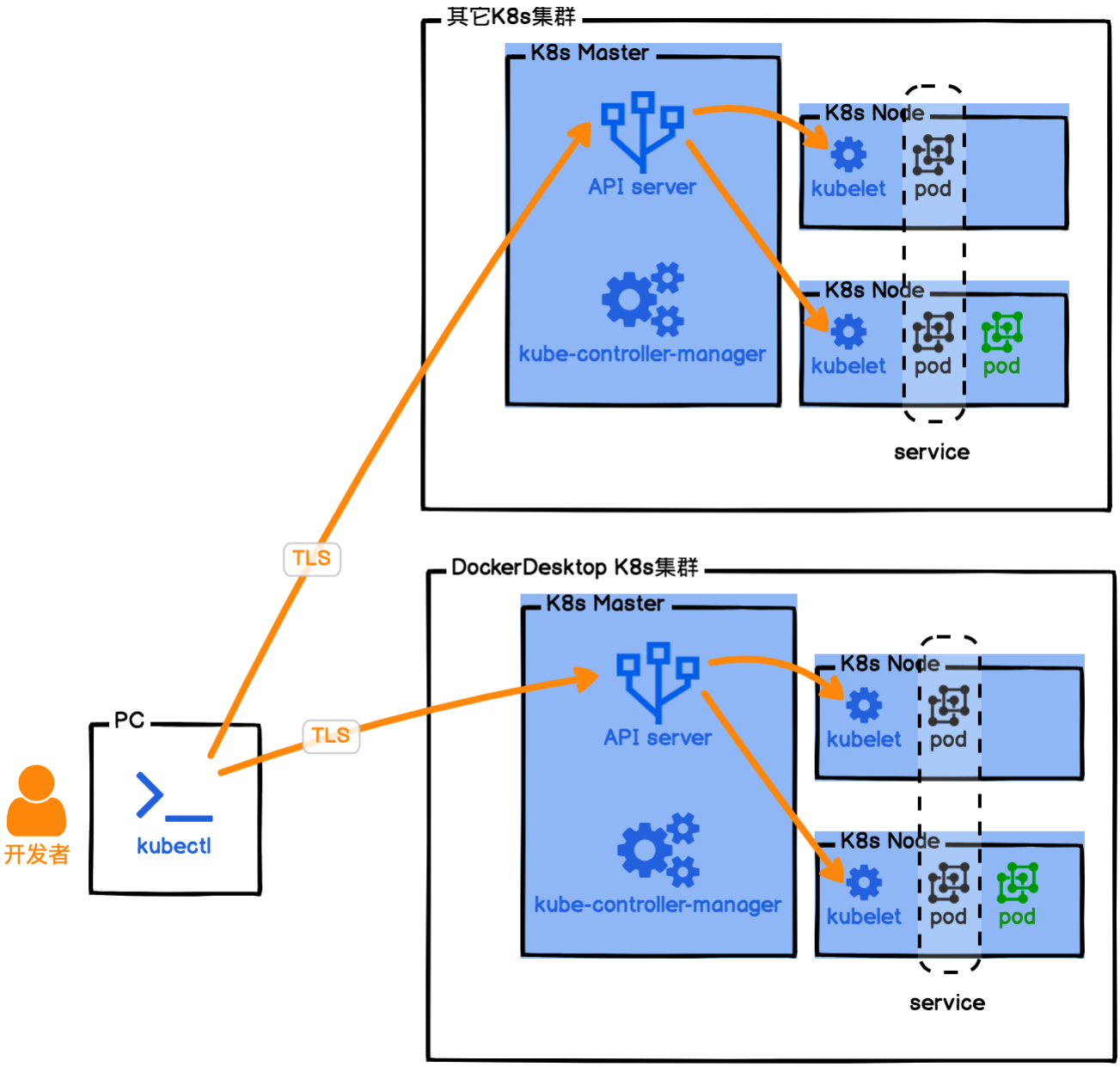
K8s介绍

安装完K8s之后，我们其实还是要简单地了解下K8s，这对于你后面应用它有重要的意义。

我想你应该知道，K8s是云原生Cloud Native[2] 的重要组成部分（目前，K8s的文档[3]已经全面中文化，建议你多翻阅），而云原生其实就是一套通用的云服务商解决方案。在我们的前几节课中，就有同学问：“如果我使用了某个运营商的Serverless，就和这个云服务商强制绑定了怎么办？我是不是就没办法使用其他运营商的服务了？”这就是服务商锁定vendor-lock，而云原生的诞生就是为了解决这个问题，通过云原生基金会CNCF(Cloud Native Computing Foundation)[4]，我们可以得到一整套解锁云服务商的开源解决方案。

那K8s作为云原生Cloud Native的重要组成部分之一，它的作用是什么呢？这里我先留一个悬念，通过理解K8s的原理，你就能清楚这个问题，并充分利用好K8s了。

我们先来看看K8s的原理图：



通过图示我们可以知道，PC本地安装kubectl是K8s的命令行操作工具，通过它，我们就可以控制K8s集群了。又因为kubectl是通过加密通信的，所以我们可以在一台电脑上同时控制多个K8s集群，不过需要指定当前操作的上下文context。这个也就是云原生的重要理念，我们的架构可以部署在多套云服务环境上。

在K8s集群中，Master节点很重要，它是我们整个集群的中枢。没错，Master节点就是Stateful的。Master节点由API Server、etcd、kube-controller-manager等几个核心成员组成，它只负责维持整个K8s集群的状态，为了保证职责单一，Master节点不会运行我们的容器实例。

Worker节点，也就是K8s Node节点，才是我们部署的容器真正运行的地方，但在K8s中，运行容器的最小单位是Pod。一个Pod具备一个集群IP且端口共享，Pod里可以运行一个或多个容器，但最佳的做法还是一个Pod只运行一个容器。这是因为一个Pod里面运行多个容器，容器会竞争Pod的资源，也会影响Pod的启动速度；而一个Pod里只运行一个容器，可以方便我们快速定位问题，监控指标也比较明确。

在K8s集群中，它会构建自己的私有网络，每个容器都有自己的集群IP，容器在集群内部可以互相访问，集群外却无法直接访问。因此我们如果要从外部访问K8s集群提供的服务，则需要通过K8s service将服务暴露出来才行。

案例：“待办任务”K8s版本

现在原理我是讲出来了，但可能还是有点不好理解，接下来我们就还是套进案例去看，依然是我们的“待办任务”Web服务，我们现在把它部署到K8s集群中运行一下，你可以切身体验。相信这样，你就非常清楚这其中的原理了。不过我们本地搭建的例子中，为了节省资源只有一个Master节点，所有的内容都部署在这个Master节点中。

还记得我们上节课构建的Docker镜像吗？我们就用它来部署本地的K8s集群。

我们通常在实际项目中会使用YAML文件来控制我们的应用部署。YAML你可以理解为，就是将我们在K8s部署的所有要做的事情，都写成一个文件，这样就避免了我们要记录大量的kubectl命令执行。不过，K8s也细心地帮我们准备了K8s对象和YAML文件互相转换的能力。这种能力可以让我们快速地将一个K8s集群中部署的结构导出YAML文件，然后再在另一个K8s集群中用这个YAML文件还原出同样的部署结构。

我们需要先确认一下，我们当前的操作是在正确的K8s集群上下文中。对应我们的例子里，也就是看当前选中的集群是不是docker-desktop。

```
kubectl config get-contexts
```

```
→  todolist-backend git:(lesson08) kubectl config get-contexts
CURRENT   NAME                 CLUSTER          AUTHINFO         NAMESPACE
*         docker-desktop       docker-desktop   docker-desktop
          docker-for-desktop   docker-desktop   docker-desktop
          minikube             minikube         minikube
```

如果不对，则需要执行切换集群：

```
kubectl config use-context docker-desktop
```

然后需要我们添加一下拉取镜像的证书服务：

```
kubectl create secret docker-registry regcred --docker-server=registry.cn-shanghai.aliyuncs.com --docker-username=你的容器镜像仓库用户名 --docker-password=你的容器镜像仓库密码
```

这里我需要解释一下，通常我们在镜像仓库中可以设置这个仓库：公开或者私有。如果是操作系统的镜像，设置为公开是完全没有问题的，所有人都可以下载我们的公开镜像；但如果是我们自己的应用镜像，还是需要设置成私有，下载私有镜像需要验证用户身份，也就是Docker Login的操作。因为我们应用的镜像仓库中，包含我们的最终运行代码，往往会有我们数据库的登录用户名和密码，或者我们云服务的ak/sk，这些重要信息如果泄露，很容易让我们的应用受到攻击。

当你添加完secret后，就可以通过下面的命令来查看secret服务了：

```
kubectl get secret regcred
```

另外我还需要再啰嗦一下，secret也不建议你用YAML文件设置，毕竟放着你用户名和密码的文件还是越少越好。

做完准备工作，对我们这次部署的项目而言就很简单了，只需要再执行一句话：

```
kubectl apply -f myapp.yaml
```

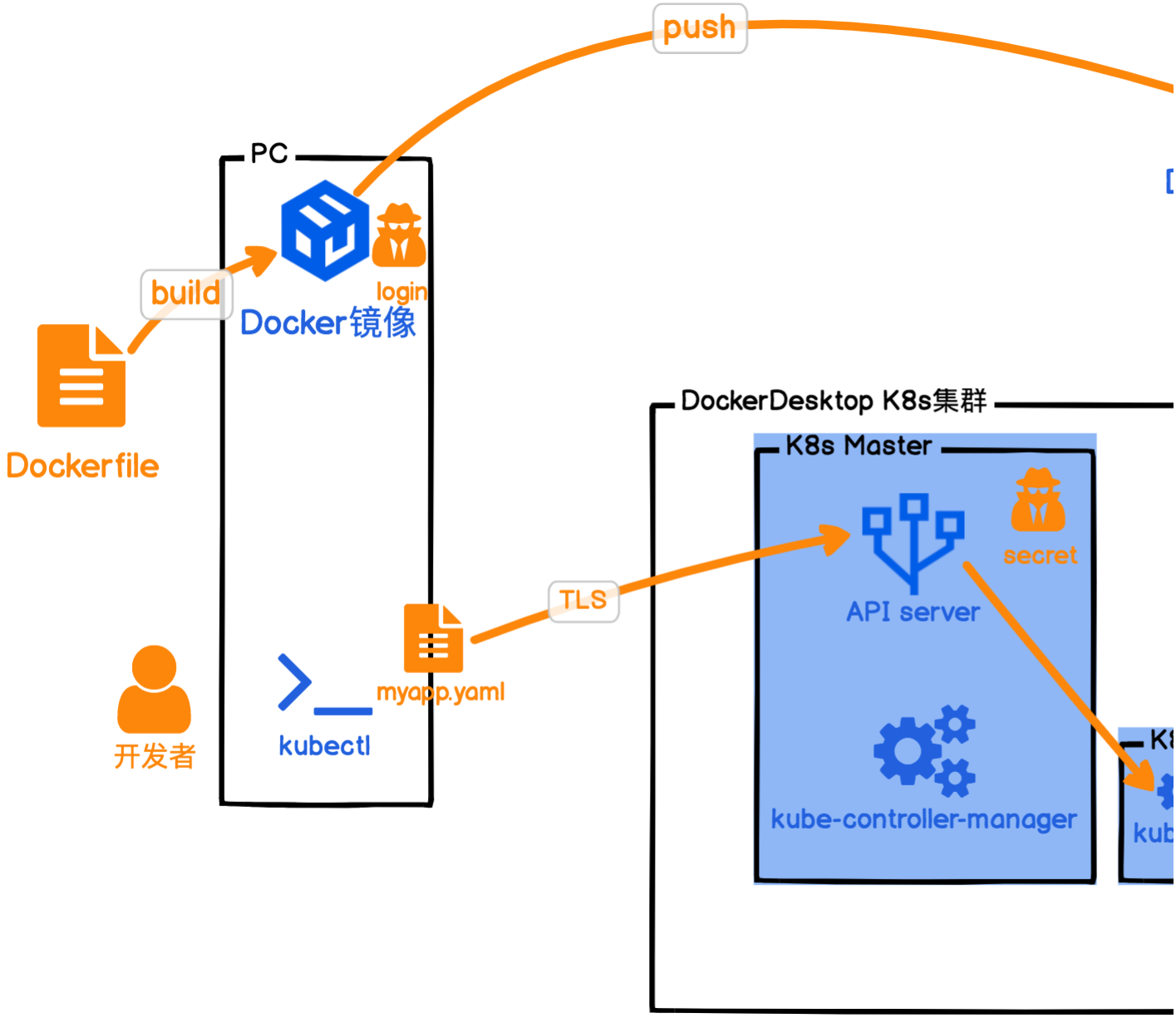
这句话的意思就是，告诉K8s集群，请按照我的部署文件myapp.yaml，部署我的应用。具体如下图所示：

```
➜  todolist-backend git:(lesson08) kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/myapp-759f79bd58-mjwwh   1/1     Running   2          21h

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          29h
service/myapp        NodePort    10.109.56.77   <none>        3001:30512/TCP   21h

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp   1/1     1            1           21h

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-759f79bd58   1         1         1       21h
```

通过获取容器的运行状态，对照上图我粗略地讲解一下我们的myapp.yaml文件吧。

- 首先我们指定要创建一个service/myapp，它选中打了"app:myapp"标签的Pod，集群内访问端口号3001，并且暴露service的端口号30512。
- 然后我们创建了部署服务deployment.apps/myapp，它负责保持我们的Pod数量恒久为1，并且给Pod打上"app:myapp"的标签，也就是负责我们的Pod持久化，一旦Pod挂了，部署服务就会重新拉起一个。
- 最后我们的容器服务，申明了自己的Docker镜像是什么，拉取镜像的secret，以及需要什么资源。

现在我们再回看K8s的原理图，不过这张是实现“待办任务”Web服务版本的：



首先我们可以看出，使用K8s仍然需要我们上节课的Docker发布流程：build、ship、run。不过现在有很多云服务商也会提供Docker镜像构建服务，你只需要上传你的Dockerfile，就会帮你构建镜像并且push到镜像仓库。云服务商提供的镜像构建服务的速度，比你本地构建要快很多倍。

而且相信你也发现了，K8s其实就是一套Docker容器实例的运行保障机制。我们自己Run一个Docker镜像，会有许多因素要考虑，例如安全性、网络隔离、日志、性能监控等等。这些K8s都帮我们考虑到了，它提供了一个Docker运行的最佳环境架构，而且还是开源的。

还有，既然我们本地都可以运行K8s的YAML文件，那么我们在云上是不是也能运行？你还记得前面我们留的悬念吧，现在就解决了。

通过K8s，我们要解开云服务商锁定vendor-lock就很简单了。我们只需要将云服务商提供的K8s环境添加到我们kubectl的配置文件中，就可以让我们的应用运行在云服务商的K8s环境中了。目前所有的较大的云服务商都已经加入CNCF，所以当你掌握K8s后，就可以根据云服务商的价格和自己喜好，自由地选择将你的K8s集群部署在CNCF成员的云服务商上，甚至你也可以在自己的机房搭建K8s环境，部署你的应用。

到这儿，我们就部署好了一个K8s集群，那之后我们该如何实时监控容器的运行状态呢？K8s既然能管理容器集群，控制容器运行实例的个数，那应该也能实时监测容器，帮我们解决扩缩容的问题吧？是的，其实上节课我们已经介绍到了容器扩缩容的原理，但并没有给你讲如何实现，那接下来我们就重点看看K8s如何实现实时监控和自动扩缩容。

K8s如何实现扩缩容？

首先，我们要知道的一点就是，K8s其实还向我们隐藏了一部分内容，就是它自身的服务状态。而我们不指定命名空间，默认的命名空间其实是default空间。要查看K8s集群系统的运行状态，我们可以通过指定namespace为kube-system来查看。K8s集群通过namespace隔离，一定程度上，隐藏了系统配置，这可以避免我们误操作。另外它也提供给我们一种逻辑隔离手段，将不同用途的服务和节点划分开来。

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          29h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          29h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          29h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/storage-provisioner                      1/1     Running   0          29h

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   29h

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns   2/2     2            2           29h

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4   2         2         2       29h
```

没错，K8s自己的服务也是运行在自己的集群中的，不过是通过命名空间，将自己做了隔离。这里需要你注意的是，这些服务我不建议你尝试去修改，因为它们涉及到了K8s集群的稳定性；但同时，K8s集群本身也具备扩展性：我们可以通过给K8s安装组件Component，扩展K8s的能力。接下来我先向你介绍K8s中的性能指标metrics组件[5]。

我的代码根目录下已经准备好了metric组件的YAML文件，你只需要执行安装就可以了：

```
kubectl apply -f metrics-components.yaml
```

安装完后，我们再看K8s的系统命名空间：

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          30h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          30h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          30h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/metrics-server-74657b4dc4-t979q          1/1     Running   0          100s
pod/storage-provisioner                      1/1     Running   0          29h

NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns         ClusterIP   10.96.0.10     <none>        53/UDP,53/TCP,9153/TCP   30h
service/metrics-server   ClusterIP   10.97.152.64   <none>        443/TCP                  100s

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns           2/2     2            2           30h
deployment.apps/metrics-server    1/1     1            1           100s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4         2         2         2       30h
replicaset.apps/metrics-server-74657b4dc4  1         1         1       100s
```

对比你就能发现，我们已经安装并启动了metrics-server。那么metrics组件有什么用呢？我们执行下面的命令看看：

```
kubectl top node
```

```
➜  todolist-backend git:(lesson08) kubectl top node
NAME             CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
docker-desktop   316m         7%     1249Mi          66%
```

安装metrics组件后，它就可以将我们应用的监控指标metrics显示出来了。没错，这里我们又可以用到上一讲的内容了。既然我们有了实时的监控指标，那么我们就可以依赖这个指标，来做我们的自动扩缩容了：

```
kubectl autoscale deployment myapp --cpu-percent=30 --min=1 --max=3
```

上面这句话的意思就是，添加一个自动扩缩容部署服务，cpu水位是30%，最小维持1个Pod，最大维持3个Pod。执行完后，我们就发现会多了一个部署服务。

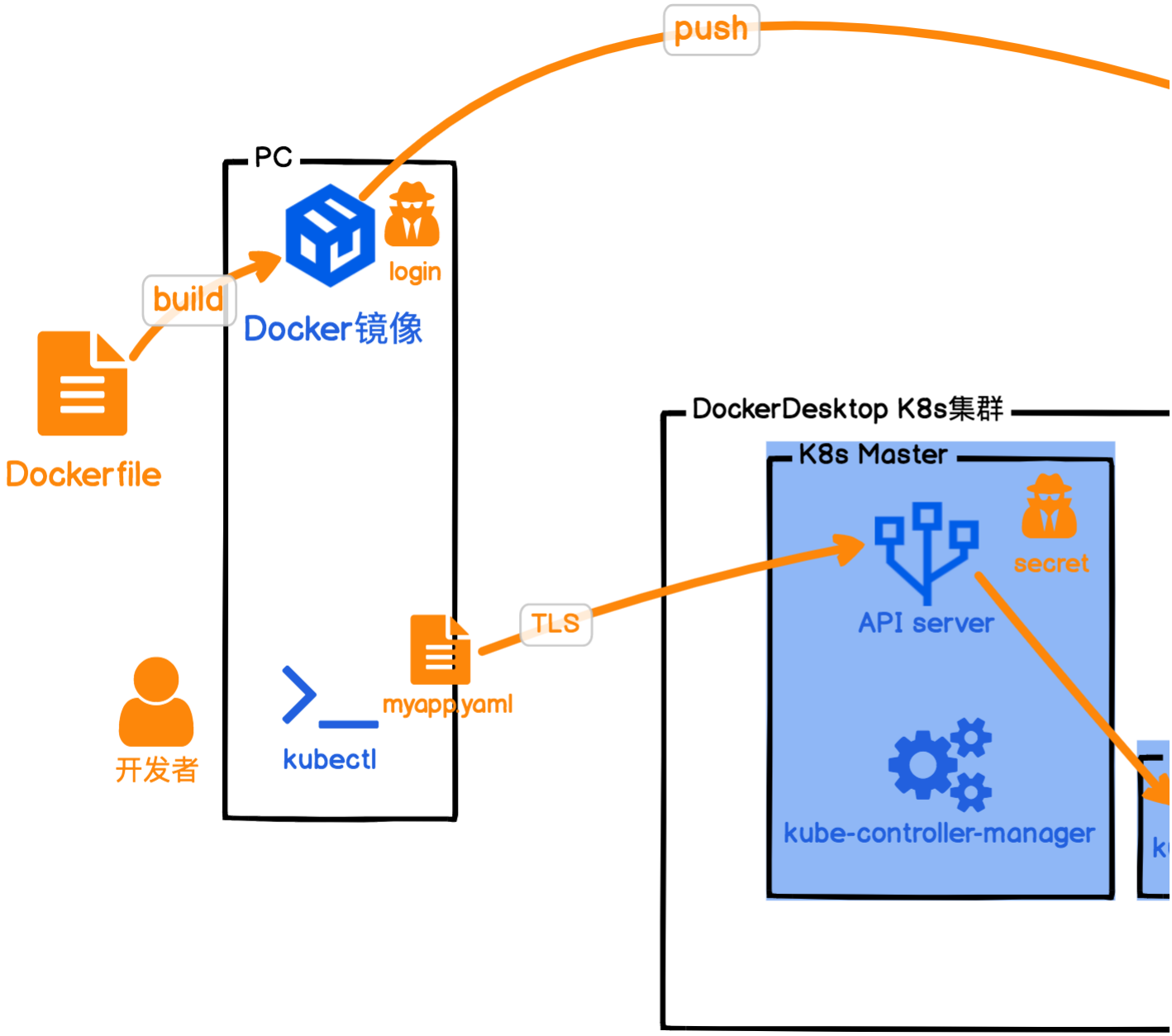
```
kubectl get hpa
```

```
→  todolist-backend git:(lesson08) kubectl get hpa
NAME     REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myapp    Deployment/myapp    0%/30%    1         3         1          22h
```

接下来，我们就可以模拟压测了：

```
kubectl run -i --tty load-generator --image=busybox /bin/sh
$ while true; do wget -q -O- http://10.1.0.16:3001/api/rule; done
```

这里我们用一个K8s的Pod，启动busybox镜像，执行死循环，压测我们的MyApp服务。不过我们目前用Node.js实现的应用可以扛住的流量比较大，单机模拟的压测，轻易还压测不到扩容的水位。



总结

这节课我向你介绍了云原生基金会CNCF的重要成员：Kubernetes。**K8s是用于自动部署、扩展和管理容器化应用程序的开源系统。**云原生其实就是一套通用的云服务商解决方案。

然后我们一起体验了在本地PC上，通过Docker desktop搭建K8s。搭建完后，我还向你介绍了K8s的运行原理：K8s Master节点和Worker节点。其中，Master节点，负责我们整个K8s集群的运作状态；Worker节点则是具体运行我们容器的地方。

之后，我们就开始把“待办任务”Web服务，通过一个K8s的YAML文件来部署，并且暴露NodePort，让我们用浏览器访问。

为了展示K8s如何通过组件Component扩展能力，接着我们介绍了K8s中如何安装使用组件metrics：我们通过一个YAML文件将metrics组件安装到了K8s集群的kube-system命名空间中后，就可以监控到应用的运行指标metrics了。给K8s集群添加上监控指标metrics的能力，我们就可以通过autoscale命令，让应用根据metrics指标和水位来扩缩容了。

最后我们启动了一个BusyBox的Docker容器，模拟压测了我们的“待办任务”Web服务。

总的来说，这节课我们的最终目的就是在本地部署一套K8s集群，通过我们“待办任务”Web服务的K8s版本，让你了解K8s的工作原理。我们已经把下节课的准备工作做好了，下节课我们将在K8s的基础上部署Serverless，可以说，实现属于你自己的Serverless，你已经完成了一半。

作业

这节课是实战课，所以作业就是我们今天要练习的内容。请在你自己的电脑上安装K8s集群，部署我们的“待办任务”Web服务到自己的K8s集群，并从浏览器中访问到K8s集群提供的服务。

另外，你可以尝试一下，手动删除我们部署的MyApp Pod。

```
kubectl delete pod/你的pod名字
```

但你很快就会发现，这个Pod会被K8s重新拉起，而我们要清除部署的MyApp的所有内容其实也很简单，只需要告诉K8s删除我们的myapp.yaml文件创建的资源就可以了。

```
kubectl delete -f myapp.yaml
```

快来动手尝试一下吧，期待你也能分享下今天的成果以及感受。另外，如果今天的内容让你有所收获，也欢迎你把它分享给身边的朋友，邀请他加入学习。

参考资料

[1] https://gitforwindows.org/

[2] https://www.cncf.io/

[3] https://kubernetes.io/zh/

[4] https://github.com/cncf/landscape

[5] https://github.com/kubernetes-incubator/metrics-server/

你好，我是秦粤。上节课我们只是用Docker部署了index.js，如果我们将所有拆解出来的微服务都用Docker独立部署，我们就要同时管理多个Docker容器，也就是Docker集群。如果是更复杂一些的业务，可能需要同时管理几十甚至上百个微服务，显然我们手动维护Docker集群的效率就太低了。而容器即服务CaaS，恰好也需要集群的管理工具。我们也知道FaaS的底层就是CaaS，那CaaS又是如何管理这么多函数实例的呢？怎么做才能提升效率？

我想你应该听过Kubernetes，它也叫K8s（后面统一简称K8s），用于自动部署、扩展和管理容器化应用程序的开源系统，是Docker集群的管理工具。为了解决上述问题，其实我们就可以考虑使用它。K8s的好处就在于，它具备跨环境统一部署的能力。

这节课，我们就试着在本地环境中搭建K8s来管理我们的Docker集群。但正常情况下，这个场景需要几台机器才能完成，而通过Docker，我们还是可以用一台机器就可以在本地搭建一个低配版的K8s。

下节课，我们还会在今天内容的基础上，用K8s的CaaS方式实现一套Serverless环境。通过这两节课的内容，你就可以完整地搭建出属于自己的Serverless了。

话不多说，我们现在就开始，希望你能跟着我一起多动手。

PC上的K8s

那在开始之前，我们先得把安装问题解决了，这部分可能会有点小困难，所以我也给你详细讲下。

首先我们需要安装kubectl，它是K8s的命令行工具。

你需要在你的PC上安装K8s，如果你的操作系统是MacOS或者Windows，那么就比较简单了，桌面版的Docker已经自带了K8s；其它操作系统的同学需要安装minikube。

不过，要顺利启动桌面版Docker自带的K8s，你还得解决国内Docker镜像下载不了的问题，这里请你先下载第8课的代码。接着，请你跟着我的步骤进行操作：

1. 开通阿里云的容器镜像仓库；
2. 在阿里云的容器镜像服务里，找到镜像加速器，复制你的镜像加速器地址；
3. 打开桌面版Docker的控制台，找到Docker Engine。

```
{
  "registry-mirrors" : [
    "https://你的加速地址.mirror.aliyuncs.com"
  ],
  "debug" : true,
  "experimental" : true
}
```

4. 预下载K8s所需要的所有镜像，执行我目录下的docker-k8s-prefetch.sh，如果你是Windows操作系统，建议使用gitBash[1]；

```
chmode +x docker-k8s-prefetch.sh
./docker-k8s-prefetch.sh
```

5. 上面拉取完运行K8s所需的Docker镜像，你就可以在桌面版Docker的K8s项中，勾选启动K8s了。

现在，K8s就能顺利启动了，启动成功后，请你继续执行下面的命令。

查看安装好的K8s系统运行状况。

```
kubectl get all -n kube-system
```

查看K8s有哪些配置环境，对应~/.kube/config。

```
kubectl config get-contexts
```

查看当前K8s环境的整体情况。

```
kubectl get all
```

按照我的流程走，在大部分的机器上本地运行K8s都是没有问题的，如果你卡住了，请在留言区告知我，我帮你解决问题。

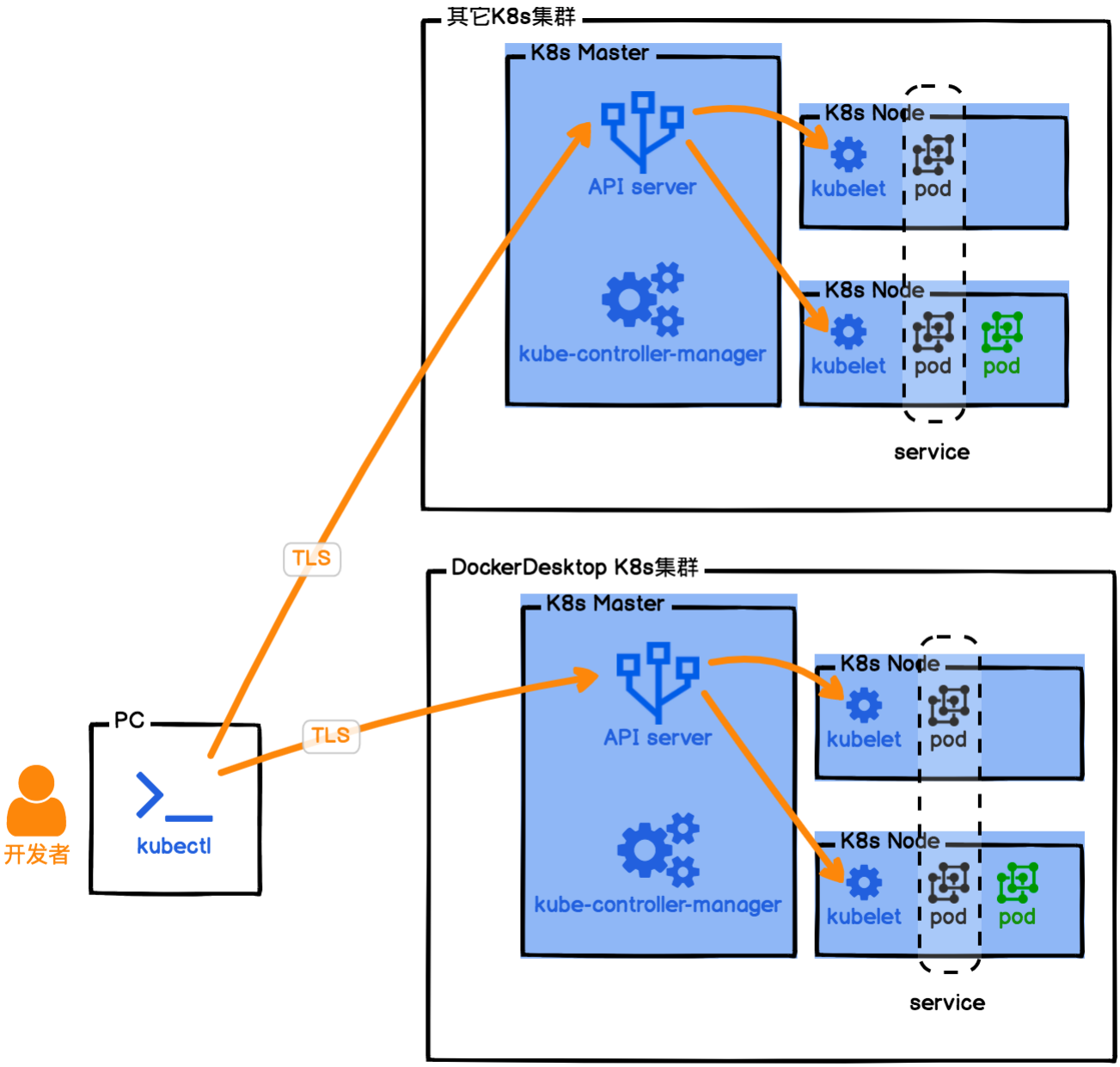
K8s介绍

安装完K8s之后，我们其实还是要简单地了解下K8s，这对于你后面应用它有重要的意义。

我想你应该知道，K8s是云原生Cloud Native[2] 的重要组成部分（目前，K8s的文档[3]已经全面中文化，建议你多翻阅），而云原生其实就是一套通用的云服务商解决方案。在我们的前几节课中，就有同学问：“如果我使用了某个运营商的Serverless，就和这个云服务商强制绑定了怎么办？我是不是就没办法使用其他运营商的服务了？”这就是服务商锁定vendor-lock，而云原生的诞生就是为了解决这个问题，通过云原生基金会CNCF(Cloud Native Computing Foundation)[4]，我们可以得到一整套解锁云服务商的开源解决方案。

那K8s作为云原生Cloud Native的重要组成部分之一，它的作用是什么呢？这里我先留一个悬念，通过理解K8s的原理，你就能清楚这个问题，并充分利用好K8s了。

我们先来看看K8s的原理图：



通过图示我们可以知道，PC本地安装kubectl是K8s的命令行操作工具，通过它，我们就可以控制K8s集群了。又因为kubectl是通过加密通信的，所以我们可以在一台电脑上同时控制多个K8s集群，不过需要指定当前操作的上下文context。这个也就是云原生的重要理念，我们的架构可以部署在多套云服务环境上。

在K8s集群中，Master节点很重要，它是我们整个集群的中枢。没错，Master节点就是Stateful的。Master节点由API Server、etcd、kube-controller-manager等几个核心成员组成，它只负责维持整个K8s集群的状态，为了保证职责单一，Master节点不会运行我们的容器实例。

Worker节点，也就是K8s Node节点，才是我们部署的容器真正运行的地方，但在K8s中，运行容器的最小单位是Pod。一个Pod具备一个集群IP且端口共享，Pod里可以运行一个或多个容器，但最佳的做法还是一个Pod只运行一个容器。这是因为一个Pod里面运行多个容器，容器会竞争Pod的资源，也会影响Pod的启动速度；而一个Pod里只运行一个容器，可以方便我们快速定位问题，监控指标也比较明确。

在K8s集群中，它会构建自己的私有网络，每个容器都有自己的集群IP，容器在集群内部可以互相访问，集群外却无法直接访问。因此我们如果要从外部访问K8s集群提供的服务，则需要通过K8s service将服务暴露出来才行。

案例：“待办任务”K8s版本

现在原理我是讲出来了，但可能还是有点不好理解，接下来我们就还是套进案例去看，依然是我们的“待办任务”Web服务，我们现在把它部署到K8s集群中运行一下，你可以切身体验。相信这样，你就非常清楚这其中的原理了。不过我们本地搭建的例子中，为了节省资源只有一个Master节点，所有的内容都部署在这个Master节点中。

还记得我们上节课构建的Docker镜像吗？我们就用它来部署本地的K8s集群。

我们通常在实际项目中会使用YAML文件来控制我们的应用部署。YAML你可以理解为，就是将我们在K8s部署的所有要做的事情，都写成一个文件，这样就避免了我们要记录大量的kubectl命令执行。不过，K8s也细心地帮我们准备了K8s对象和YAML文件互相转换的能力。这种能力可以让我们快速地将一个K8s集群中部署的结构导出YAML文件，然后再在另一个K8s集群中用这个YAML文件还原出同样的部署结构。

我们需要先确认一下，我们当前的操作是在正确的K8s集群上下文中。对应我们的例子里，也就是看当前选中的集群是不是docker-desktop。

```
kubectl config get-contexts
```

```
→  todolist-backend git:(lesson08) kubectl config get-contexts
CURRENT   NAME                 CLUSTER          AUTHINFO         NAMESPACE
*         docker-desktop       docker-desktop   docker-desktop
          docker-for-desktop   docker-desktop   docker-desktop
          minikube             minikube         minikube
```

如果不对，则需要执行切换集群：

```
kubectl config use-context docker-desktop
```

然后需要我们添加一下拉取镜像的证书服务：

```
kubectl create secret docker-registry regcred --docker-server=registry.cn-shanghai.aliyuncs.com --docker-username=你的容器镜像仓库用户名 --docker-password=你的容器镜像仓库密码
```

这里我需要解释一下，通常我们在镜像仓库中可以设置这个仓库：公开或者私有。如果是操作系统的镜像，设置为公开是完全没有问题的，所有人都可以下载我们的公开镜像；但如果是我们自己的应用镜像，还是需要设置成私有，下载私有镜像需要验证用户身份，也就是Docker Login的操作。因为我们应用的镜像仓库中，包含我们的最终运行代码，往往会有我们数据库的登录用户名和密码，或者我们云服务的ak/sk，这些重要信息如果泄露，很容易让我们的应用受到攻击。

当你添加完secret后，就可以通过下面的命令来查看secret服务了：

```
kubectl get secret regcred
```

另外我还需要再啰嗦一下，secret也不建议你用YAML文件设置，毕竟放着你用户名和密码的文件还是越少越好。

做完准备工作，对我们这次部署的项目而言就很简单了，只需要再执行一句话：

```
kubectl apply -f myapp.yaml
```

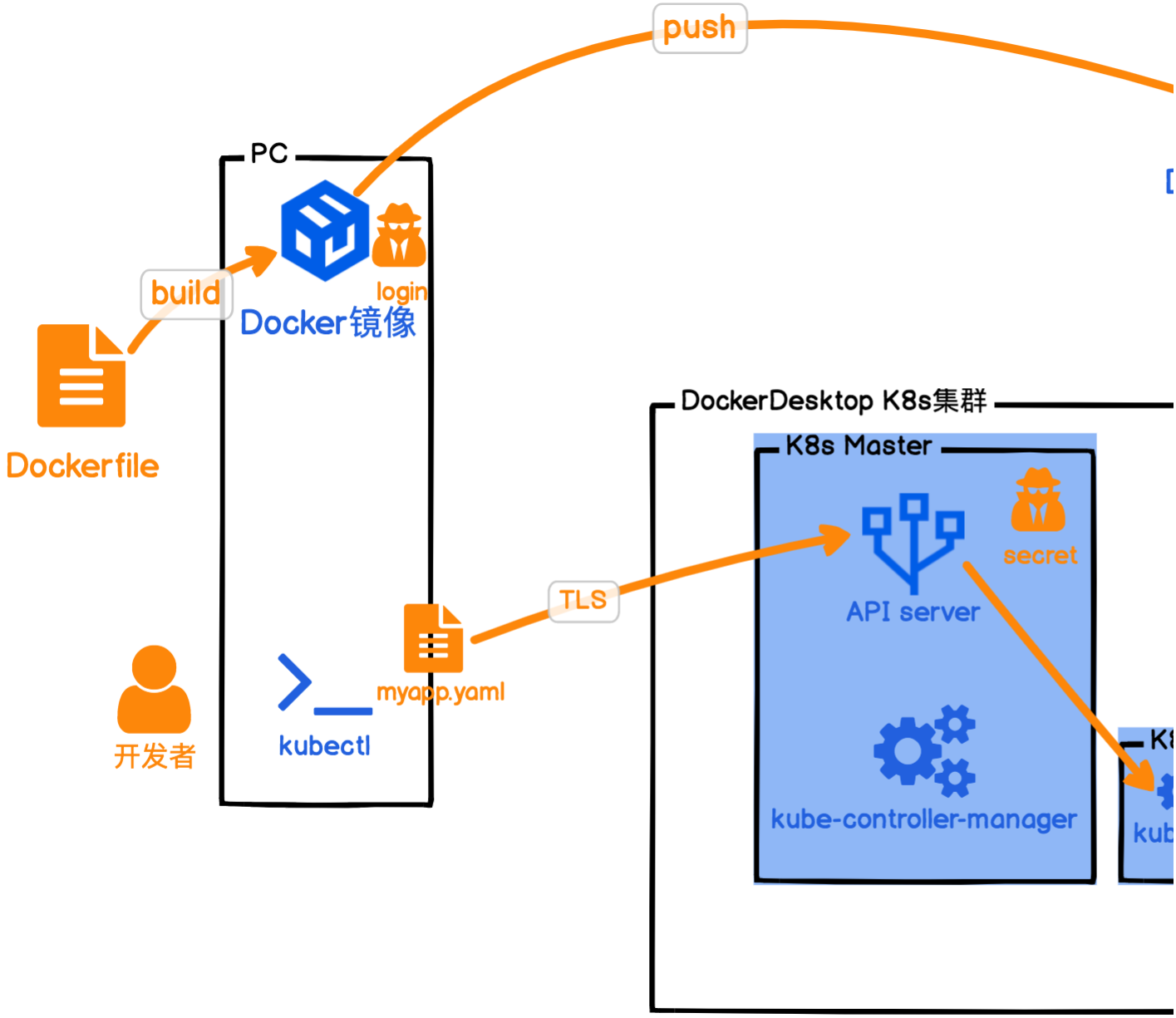
这句话的意思就是，告诉K8s集群，请按照我的部署文件myapp.yaml，部署我的应用。具体如下图所示：

```
➜  todolist-backend git:(lesson08) kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/myapp-759f79bd58-mjwwh   1/1     Running   2          21h

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          29h
service/myapp        NodePort    10.109.56.77   <none>        3001:30512/TCP   21h

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp   1/1     1            1           21h

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-759f79bd58   1         1         1       21h
```

通过获取容器的运行状态，对照上图我粗略地讲解一下我们的myapp.yaml文件吧。

- 首先我们指定要创建一个service/myapp，它选中打了"app:myapp"标签的Pod，集群内访问端口号3001，并且暴露service的端口号30512。
- 然后我们创建了部署服务deployment.apps/myapp，它负责保持我们的Pod数量恒久为1，并且给Pod打上"app:myapp"的标签，也就是负责我们的Pod持久化，一旦Pod挂了，部署服务就会重新拉起一个。
- 最后我们的容器服务，申明了自己的Docker镜像是什么，拉取镜像的secret，以及需要什么资源。

现在我们再回看K8s的原理图，不过这张是实现“待办任务”Web服务版本的：



首先我们可以看出，使用K8s仍然需要我们上节课的Docker发布流程：build、ship、run。不过现在有很多云服务商也会提供Docker镜像构建服务，你只需要上传你的Dockerfile，就会帮你构建镜像并且push到镜像仓库。云服务商提供的镜像构建服务的速度，比你本地构建要快很多倍。

而且相信你也发现了，K8s其实就是一套Docker容器实例的运行保障机制。我们自己Run一个Docker镜像，会有许多因素要考虑，例如安全性、网络隔离、日志、性能监控等等。这些K8s都帮我们考虑到了，它提供了一个Docker运行的最佳环境架构，而且还是开源的。

还有，既然我们本地都可以运行K8s的YAML文件，那么我们在云上是不是也能运行？你还记得前面我们留的悬念吧，现在就解决了。

通过K8s，我们要解开云服务商锁定vendor-lock就很简单了。我们只需要将云服务商提供的K8s环境添加到我们kubectl的配置文件中，就可以让我们的应用运行在云服务商的K8s环境中了。目前所有的较大的云服务商都已经加入CNCF，所以当你掌握K8s后，就可以根据云服务商的价格和自己喜好，自由地选择将你的K8s集群部署在CNCF成员的云服务商上，甚至你也可以在自己的机房搭建K8s环境，部署你的应用。

到这儿，我们就部署好了一个K8s集群，那之后我们该如何实时监控容器的运行状态呢？K8s既然能管理容器集群，控制容器运行实例的个数，那应该也能实时监测容器，帮我们解决扩缩容的问题吧？是的，其实上节课我们已经介绍到了容器扩缩容的原理，但并没有给你讲如何实现，那接下来我们就重点看看K8s如何实现实时监控和自动扩缩容。

K8s如何实现扩缩容？

首先，我们要知道的一点就是，K8s其实还向我们隐藏了一部分内容，就是它自身的服务状态。而我们不指定命名空间，默认的命名空间其实是default空间。要查看K8s集群系统的运行状态，我们可以通过指定namespace为kube-system来查看。K8s集群通过namespace隔离，一定程度上，隐藏了系统配置，这可以避免我们误操作。另外它也提供给我们一种逻辑隔离手段，将不同用途的服务和节点划分开来。

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          29h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          29h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          29h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/storage-provisioner                      1/1     Running   0          29h

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   29h

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns   2/2     2            2           29h

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4   2         2         2       29h
```

没错，K8s自己的服务也是运行在自己的集群中的，不过是通过命名空间，将自己做了隔离。这里需要你注意的是，这些服务我不建议你尝试去修改，因为它们涉及到了K8s集群的稳定性；但同时，K8s集群本身也具备扩展性：我们可以通过给K8s安装组件Component，扩展K8s的能力。接下来我先向你介绍K8s中的性能指标metrics组件[5]。

我的代码根目录下已经准备好了metric组件的YAML文件，你只需要执行安装就可以了：

```
kubectl apply -f metrics-components.yaml
```

安装完后，我们再看K8s的系统命名空间：

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          30h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          30h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          30h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/metrics-server-74657b4dc4-t979q          1/1     Running   0          100s
pod/storage-provisioner                      1/1     Running   0          29h

NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns         ClusterIP   10.96.0.10     <none>        53/UDP,53/TCP,9153/TCP   30h
service/metrics-server   ClusterIP   10.97.152.64   <none>        443/TCP                  100s

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns           2/2     2            2           30h
deployment.apps/metrics-server    1/1     1            1           100s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4         2         2         2       30h
replicaset.apps/metrics-server-74657b4dc4  1         1         1       100s
```

对比你就能发现，我们已经安装并启动了metrics-server。那么metrics组件有什么用呢？我们执行下面的命令看看：

```
kubectl top node
```

```
➜  todolist-backend git:(lesson08) kubectl top node
NAME             CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
docker-desktop   316m         7%     1249Mi          66%
```

安装metrics组件后，它就可以将我们应用的监控指标metrics显示出来了。没错，这里我们又可以用到上一讲的内容了。既然我们有了实时的监控指标，那么我们就可以依赖这个指标，来做我们的自动扩缩容了：

```
kubectl autoscale deployment myapp --cpu-percent=30 --min=1 --max=3
```

上面这句话的意思就是，添加一个自动扩缩容部署服务，cpu水位是30%，最小维持1个Pod，最大维持3个Pod。执行完后，我们就发现会多了一个部署服务。

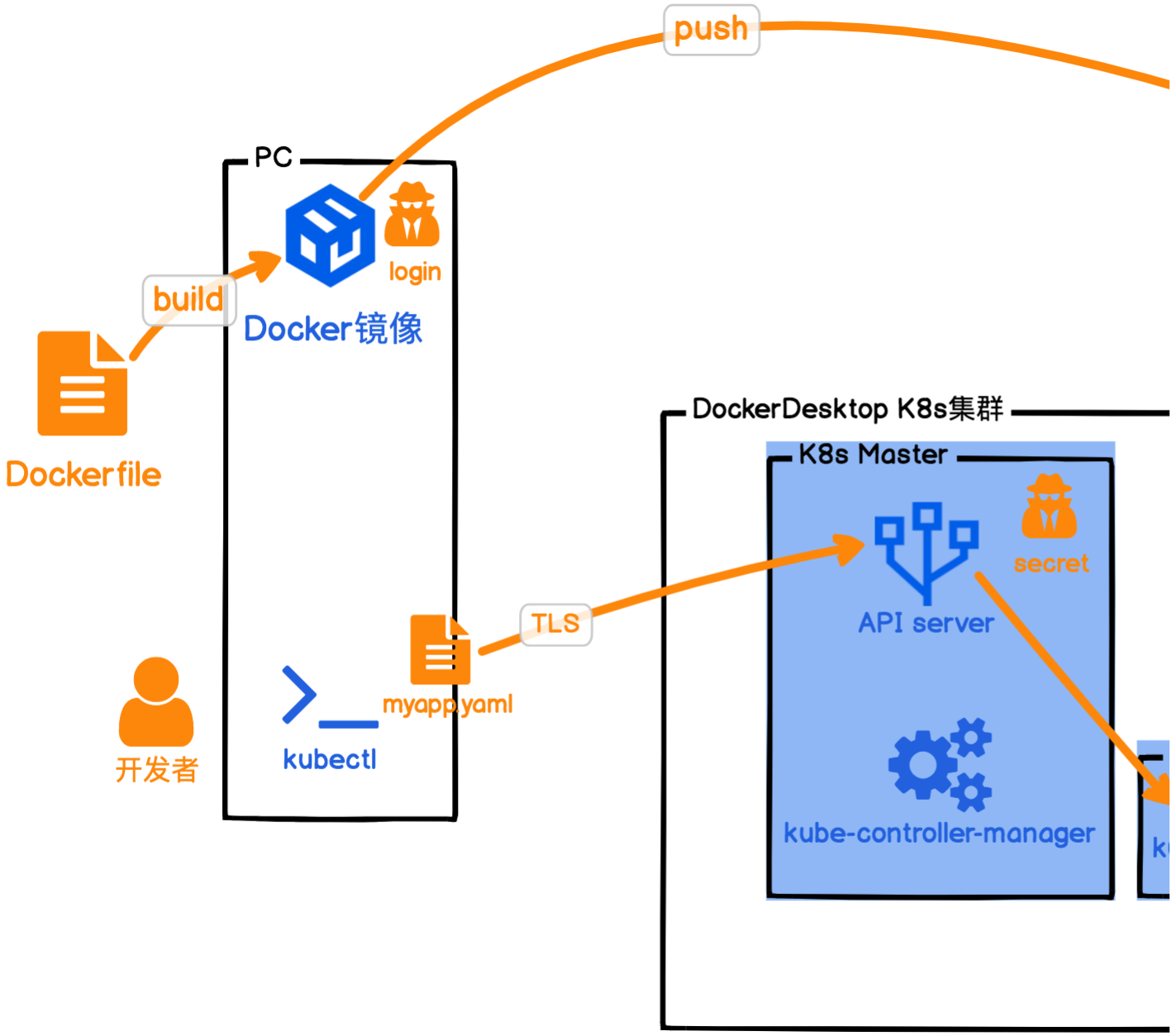
```
kubectl get hpa
```

```
→  todolist-backend git:(lesson08) kubectl get hpa
NAME     REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myapp    Deployment/myapp    0%/30%    1         3         1          22h
```

接下来，我们就可以模拟压测了：

```
kubectl run -i --tty load-generator --image=busybox /bin/sh
$ while true; do wget -q -O- http://10.1.0.16:3001/api/rule; done
```

这里我们用一个K8s的Pod，启动busybox镜像，执行死循环，压测我们的MyApp服务。不过我们目前用Node.js实现的应用可以扛住的流量比较大，单机模拟的压测，轻易还压测不到扩容的水位。



总结

这节课我向你介绍了云原生基金会CNCF的重要成员：Kubernetes。**K8s是用于自动部署、扩展和管理容器化应用程序的开源系统。**云原生其实就是一套通用的云服务商解决方案。

然后我们一起体验了在本地PC上，通过Docker desktop搭建K8s。搭建完后，我还向你介绍了K8s的运行原理：K8s Master节点和Worker节点。其中，Master节点，负责我们整个K8s集群的运作状态；Worker节点则是具体运行我们容器的地方。

之后，我们就开始把“待办任务”Web服务，通过一个K8s的YAML文件来部署，并且暴露NodePort，让我们用浏览器访问。

为了展示K8s如何通过组件Component扩展能力，接着我们介绍了K8s中如何安装使用组件metrics：我们通过一个YAML文件将metrics组件安装到了K8s集群的kube-system命名空间中后，就可以监控到应用的运行指标metrics了。给K8s集群添加上监控指标metrics的能力，我们就可以通过autoscale命令，让应用根据metrics指标和水位来扩缩容了。

最后我们启动了一个BusyBox的Docker容器，模拟压测了我们的“待办任务”Web服务。

总的来说，这节课我们的最终目的就是在本地部署一套K8s集群，通过我们“待办任务”Web服务的K8s版本，让你了解K8s的工作原理。我们已经把下节课的准备工作做好了，下节课我们将在K8s的基础上部署Serverless，可以说，实现属于你自己的Serverless，你已经完成了一半。

作业

这节课是实战课，所以作业就是我们今天要练习的内容。请在你自己的电脑上安装K8s集群，部署我们的“待办任务”Web服务到自己的K8s集群，并从浏览器中访问到K8s集群提供的服务。

另外，你可以尝试一下，手动删除我们部署的MyApp Pod。

```
kubectl delete pod/你的pod名字
```

但你很快就会发现，这个Pod会被K8s重新拉起，而我们要清除部署的MyApp的所有内容其实也很简单，只需要告诉K8s删除我们的myapp.yaml文件创建的资源就可以了。

```
kubectl delete -f myapp.yaml
```

快来动手尝试一下吧，期待你也能分享下今天的成果以及感受。另外，如果今天的内容让你有所收获，也欢迎你把它分享给身边的朋友，邀请他加入学习。

参考资料

[1] https://gitforwindows.org/

[2] https://www.cncf.io/

[3] https://kubernetes.io/zh/

[4] https://github.com/cncf/landscape

[5] https://github.com/kubernetes-incubator/metrics-server/

你好，我是秦粤。上节课我们只是用Docker部署了index.js，如果我们将所有拆解出来的微服务都用Docker独立部署，我们就要同时管理多个Docker容器，也就是Docker集群。如果是更复杂一些的业务，可能需要同时管理几十甚至上百个微服务，显然我们手动维护Docker集群的效率就太低了。而容器即服务CaaS，恰好也需要集群的管理工具。我们也知道FaaS的底层就是CaaS，那CaaS又是如何管理这么多函数实例的呢？怎么做才能提升效率？

我想你应该听过Kubernetes，它也叫K8s（后面统一简称K8s），用于自动部署、扩展和管理容器化应用程序的开源系统，是Docker集群的管理工具。为了解决上述问题，其实我们就可以考虑使用它。K8s的好处就在于，它具备跨环境统一部署的能力。

这节课，我们就试着在本地环境中搭建K8s来管理我们的Docker集群。但正常情况下，这个场景需要几台机器才能完成，而通过Docker，我们还是可以用一台机器就可以在本地搭建一个低配版的K8s。

下节课，我们还会在今天内容的基础上，用K8s的CaaS方式实现一套Serverless环境。通过这两节课的内容，你就可以完整地搭建出属于自己的Serverless了。

话不多说，我们现在就开始，希望你能跟着我一起多动手。

PC上的K8s

那在开始之前，我们先得把安装问题解决了，这部分可能会有点小困难，所以我也给你详细讲下。

首先我们需要安装kubectl，它是K8s的命令行工具。

你需要在你的PC上安装K8s，如果你的操作系统是MacOS或者Windows，那么就比较简单了，桌面版的Docker已经自带了K8s；其它操作系统的同学需要安装minikube。

不过，要顺利启动桌面版Docker自带的K8s，你还得解决国内Docker镜像下载不了的问题，这里请你先下载第8课的代码。接着，请你跟着我的步骤进行操作：

1. 开通阿里云的容器镜像仓库；
2. 在阿里云的容器镜像服务里，找到镜像加速器，复制你的镜像加速器地址；
3. 打开桌面版Docker的控制台，找到Docker Engine。

```
{
  "registry-mirrors" : [
    "https://你的加速地址.mirror.aliyuncs.com"
  ],
  "debug" : true,
  "experimental" : true
}
```

4. 预下载K8s所需要的所有镜像，执行我目录下的docker-k8s-prefetch.sh，如果你是Windows操作系统，建议使用gitBash[1]；

```
chmode +x docker-k8s-prefetch.sh
./docker-k8s-prefetch.sh
```

5. 上面拉取完运行K8s所需的Docker镜像，你就可以在桌面版Docker的K8s项中，勾选启动K8s了。

现在，K8s就能顺利启动了，启动成功后，请你继续执行下面的命令。

查看安装好的K8s系统运行状况。

```
kubectl get all -n kube-system
```

查看K8s有哪些配置环境，对应~/.kube/config。

```
kubectl config get-contexts
```

查看当前K8s环境的整体情况。

```
kubectl get all
```

按照我的流程走，在大部分的机器上本地运行K8s都是没有问题的，如果你卡住了，请在留言区告知我，我帮你解决问题。

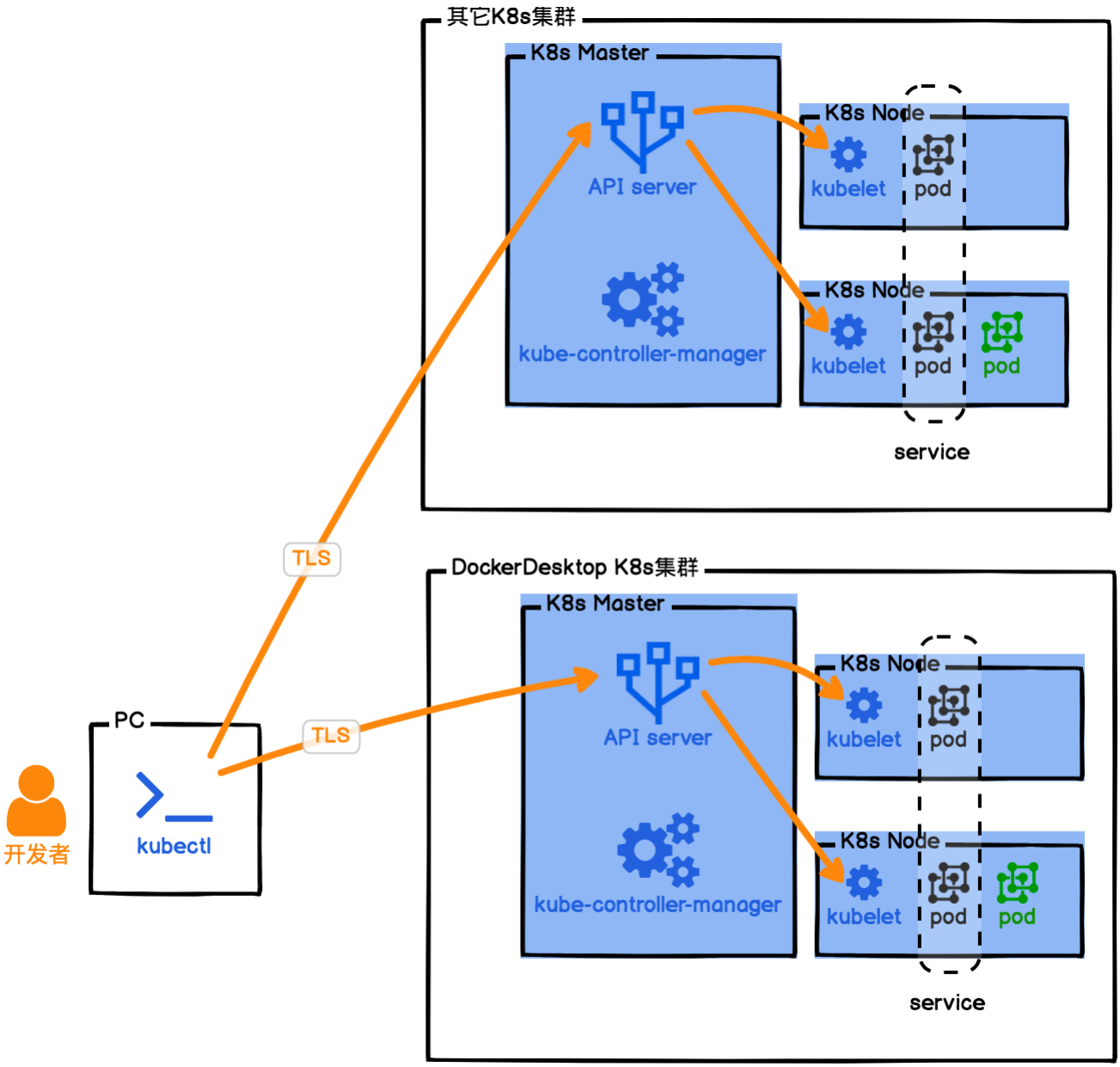
K8s介绍

安装完K8s之后，我们其实还是要简单地了解下K8s，这对于你后面应用它有重要的意义。

我想你应该知道，K8s是云原生Cloud Native[2] 的重要组成部分（目前，K8s的文档[3]已经全面中文化，建议你多翻阅），而云原生其实就是一套通用的云服务商解决方案。在我们的前几节课中，就有同学问：“如果我使用了某个运营商的Serverless，就和这个云服务商强制绑定了怎么办？我是不是就没办法使用其他运营商的服务了？”这就是服务商锁定vendor-lock，而云原生的诞生就是为了解决这个问题，通过云原生基金会CNCF(Cloud Native Computing Foundation)[4]，我们可以得到一整套解锁云服务商的开源解决方案。

那K8s作为云原生Cloud Native的重要组成部分之一，它的作用是什么呢？这里我先留一个悬念，通过理解K8s的原理，你就能清楚这个问题，并充分利用好K8s了。

我们先来看看K8s的原理图：



通过图示我们可以知道，PC本地安装kubectl是K8s的命令行操作工具，通过它，我们就可以控制K8s集群了。又因为kubectl是通过加密通信的，所以我们可以在一台电脑上同时控制多个K8s集群，不过需要指定当前操作的上下文context。这个也就是云原生的重要理念，我们的架构可以部署在多套云服务环境上。

在K8s集群中，Master节点很重要，它是我们整个集群的中枢。没错，Master节点就是Stateful的。Master节点由API Server、etcd、kube-controller-manager等几个核心成员组成，它只负责维持整个K8s集群的状态，为了保证职责单一，Master节点不会运行我们的容器实例。

Worker节点，也就是K8s Node节点，才是我们部署的容器真正运行的地方，但在K8s中，运行容器的最小单位是Pod。一个Pod具备一个集群IP且端口共享，Pod里可以运行一个或多个容器，但最佳的做法还是一个Pod只运行一个容器。这是因为一个Pod里面运行多个容器，容器会竞争Pod的资源，也会影响Pod的启动速度；而一个Pod里只运行一个容器，可以方便我们快速定位问题，监控指标也比较明确。

在K8s集群中，它会构建自己的私有网络，每个容器都有自己的集群IP，容器在集群内部可以互相访问，集群外却无法直接访问。因此我们如果要从外部访问K8s集群提供的服务，则需要通过K8s service将服务暴露出来才行。

案例：“待办任务”K8s版本

现在原理我是讲出来了，但可能还是有点不好理解，接下来我们就还是套进案例去看，依然是我们的“待办任务”Web服务，我们现在把它部署到K8s集群中运行一下，你可以切身体验。相信这样，你就非常清楚这其中的原理了。不过我们本地搭建的例子中，为了节省资源只有一个Master节点，所有的内容都部署在这个Master节点中。

还记得我们上节课构建的Docker镜像吗？我们就用它来部署本地的K8s集群。

我们通常在实际项目中会使用YAML文件来控制我们的应用部署。YAML你可以理解为，就是将我们在K8s部署的所有要做的事情，都写成一个文件，这样就避免了我们要记录大量的kubectl命令执行。不过，K8s也细心地帮我们准备了K8s对象和YAML文件互相转换的能力。这种能力可以让我们快速地将一个K8s集群中部署的结构导出YAML文件，然后再在另一个K8s集群中用这个YAML文件还原出同样的部署结构。

我们需要先确认一下，我们当前的操作是在正确的K8s集群上下文中。对应我们的例子里，也就是看当前选中的集群是不是docker-desktop。

```
kubectl config get-contexts
```

```
→  todolist-backend git:(lesson08) kubectl config get-contexts
CURRENT   NAME                 CLUSTER          AUTHINFO         NAMESPACE
*         docker-desktop       docker-desktop   docker-desktop
          docker-for-desktop   docker-desktop   docker-desktop
          minikube             minikube         minikube
```

如果不对，则需要执行切换集群：

```
kubectl config use-context docker-desktop
```

然后需要我们添加一下拉取镜像的证书服务：

```
kubectl create secret docker-registry regcred --docker-server=registry.cn-shanghai.aliyuncs.com --docker-username=你的容器镜像仓库用户名 --docker-password=你的容器镜像仓库密码
```

这里我需要解释一下，通常我们在镜像仓库中可以设置这个仓库：公开或者私有。如果是操作系统的镜像，设置为公开是完全没有问题的，所有人都可以下载我们的公开镜像；但如果是我们自己的应用镜像，还是需要设置成私有，下载私有镜像需要验证用户身份，也就是Docker Login的操作。因为我们应用的镜像仓库中，包含我们的最终运行代码，往往会有我们数据库的登录用户名和密码，或者我们云服务的ak/sk，这些重要信息如果泄露，很容易让我们的应用受到攻击。

当你添加完secret后，就可以通过下面的命令来查看secret服务了：

```
kubectl get secret regcred
```

另外我还需要再啰嗦一下，secret也不建议你用YAML文件设置，毕竟放着你用户名和密码的文件还是越少越好。

做完准备工作，对我们这次部署的项目而言就很简单了，只需要再执行一句话：

```
kubectl apply -f myapp.yaml
```

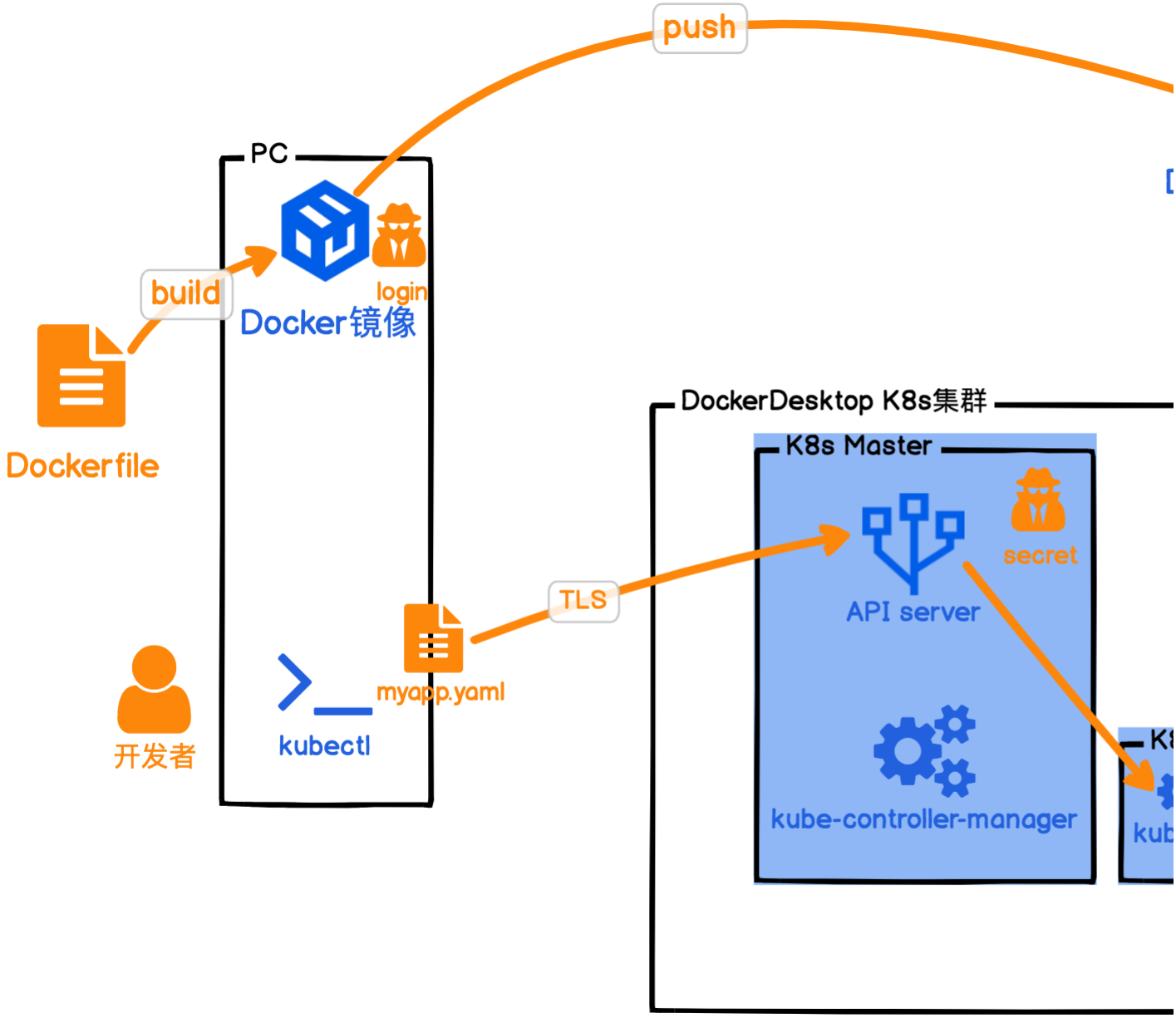
这句话的意思就是，告诉K8s集群，请按照我的部署文件myapp.yaml，部署我的应用。具体如下图所示：

```
➜  todolist-backend git:(lesson08) kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/myapp-759f79bd58-mjwwh   1/1     Running   2          21h

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          29h
service/myapp        NodePort    10.109.56.77   <none>        3001:30512/TCP   21h

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp   1/1     1            1           21h

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-759f79bd58   1         1         1       21h
```

通过获取容器的运行状态，对照上图我粗略地讲解一下我们的myapp.yaml文件吧。

- 首先我们指定要创建一个service/myapp，它选中打了"app:myapp"标签的Pod，集群内访问端口号3001，并且暴露service的端口号30512。
- 然后我们创建了部署服务deployment.apps/myapp，它负责保持我们的Pod数量恒久为1，并且给Pod打上"app:myapp"的标签，也就是负责我们的Pod持久化，一旦Pod挂了，部署服务就会重新拉起一个。
- 最后我们的容器服务，申明了自己的Docker镜像是什么，拉取镜像的secret，以及需要什么资源。

现在我们再回看K8s的原理图，不过这张是实现“待办任务”Web服务版本的：



首先我们可以看出，使用K8s仍然需要我们上节课的Docker发布流程：build、ship、run。不过现在有很多云服务商也会提供Docker镜像构建服务，你只需要上传你的Dockerfile，就会帮你构建镜像并且push到镜像仓库。云服务商提供的镜像构建服务的速度，比你本地构建要快很多倍。

而且相信你也发现了，K8s其实就是一套Docker容器实例的运行保障机制。我们自己Run一个Docker镜像，会有许多因素要考虑，例如安全性、网络隔离、日志、性能监控等等。这些K8s都帮我们考虑到了，它提供了一个Docker运行的最佳环境架构，而且还是开源的。

还有，既然我们本地都可以运行K8s的YAML文件，那么我们在云上是不是也能运行？你还记得前面我们留的悬念吧，现在就解决了。

通过K8s，我们要解开云服务商锁定vendor-lock就很简单了。我们只需要将云服务商提供的K8s环境添加到我们kubectl的配置文件中，就可以让我们的应用运行在云服务商的K8s环境中了。目前所有的较大的云服务商都已经加入CNCF，所以当你掌握K8s后，就可以根据云服务商的价格和自己喜好，自由地选择将你的K8s集群部署在CNCF成员的云服务商上，甚至你也可以在自己的机房搭建K8s环境，部署你的应用。

到这儿，我们就部署好了一个K8s集群，那之后我们该如何实时监控容器的运行状态呢？K8s既然能管理容器集群，控制容器运行实例的个数，那应该也能实时监测容器，帮我们解决扩缩容的问题吧？是的，其实上节课我们已经介绍到了容器扩缩容的原理，但并没有给你讲如何实现，那接下来我们就重点看看K8s如何实现实时监控和自动扩缩容。

K8s如何实现扩缩容？

首先，我们要知道的一点就是，K8s其实还向我们隐藏了一部分内容，就是它自身的服务状态。而我们不指定命名空间，默认的命名空间其实是default空间。要查看K8s集群系统的运行状态，我们可以通过指定namespace为kube-system来查看。K8s集群通过namespace隔离，一定程度上，隐藏了系统配置，这可以避免我们误操作。另外它也提供给我们一种逻辑隔离手段，将不同用途的服务和节点划分开来。

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          29h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          29h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          29h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/storage-provisioner                      1/1     Running   0          29h

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   29h

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns   2/2     2            2           29h

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4   2         2         2       29h
```

没错，K8s自己的服务也是运行在自己的集群中的，不过是通过命名空间，将自己做了隔离。这里需要你注意的是，这些服务我不建议你尝试去修改，因为它们涉及到了K8s集群的稳定性；但同时，K8s集群本身也具备扩展性：我们可以通过给K8s安装组件Component，扩展K8s的能力。接下来我先向你介绍K8s中的性能指标metrics组件[5]。

我的代码根目录下已经准备好了metric组件的YAML文件，你只需要执行安装就可以了：

```
kubectl apply -f metrics-components.yaml
```

安装完后，我们再看K8s的系统命名空间：

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          30h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          30h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          30h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/metrics-server-74657b4dc4-t979q          1/1     Running   0          100s
pod/storage-provisioner                      1/1     Running   0          29h

NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns         ClusterIP   10.96.0.10     <none>        53/UDP,53/TCP,9153/TCP   30h
service/metrics-server   ClusterIP   10.97.152.64   <none>        443/TCP                  100s

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns           2/2     2            2           30h
deployment.apps/metrics-server    1/1     1            1           100s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4         2         2         2       30h
replicaset.apps/metrics-server-74657b4dc4  1         1         1       100s
```

对比你就能发现，我们已经安装并启动了metrics-server。那么metrics组件有什么用呢？我们执行下面的命令看看：

```
kubectl top node
```

```
➜  todolist-backend git:(lesson08) kubectl top node
NAME             CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
docker-desktop   316m         7%     1249Mi          66%
```

安装metrics组件后，它就可以将我们应用的监控指标metrics显示出来了。没错，这里我们又可以用到上一讲的内容了。既然我们有了实时的监控指标，那么我们就可以依赖这个指标，来做我们的自动扩缩容了：

```
kubectl autoscale deployment myapp --cpu-percent=30 --min=1 --max=3
```

上面这句话的意思就是，添加一个自动扩缩容部署服务，cpu水位是30%，最小维持1个Pod，最大维持3个Pod。执行完后，我们就发现会多了一个部署服务。

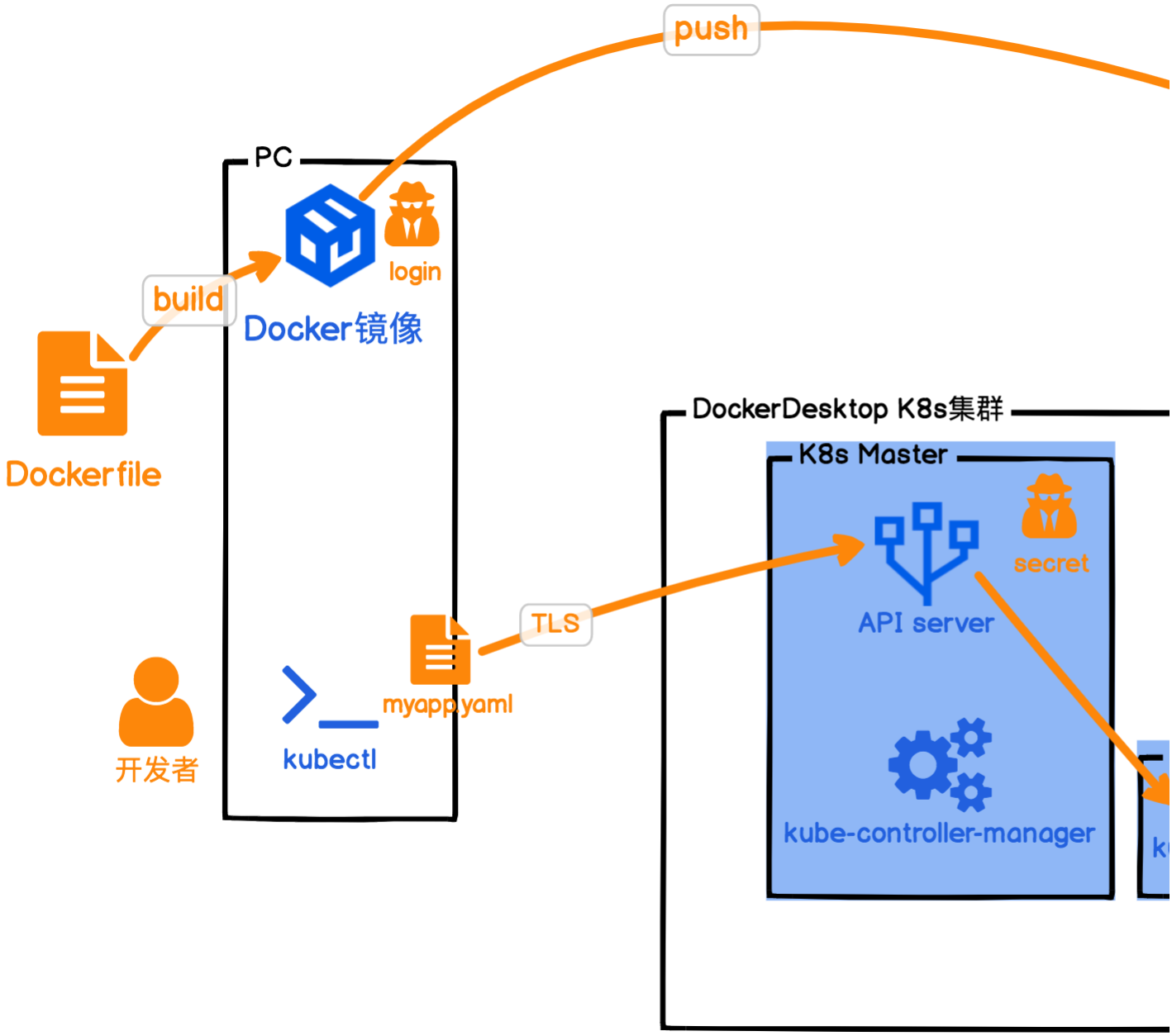
```
kubectl get hpa
```

```
→  todolist-backend git:(lesson08) kubectl get hpa
NAME     REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myapp    Deployment/myapp    0%/30%    1         3         1          22h
```

接下来，我们就可以模拟压测了：

```
kubectl run -i --tty load-generator --image=busybox /bin/sh
$ while true; do wget -q -O- http://10.1.0.16:3001/api/rule; done
```

这里我们用一个K8s的Pod，启动busybox镜像，执行死循环，压测我们的MyApp服务。不过我们目前用Node.js实现的应用可以扛住的流量比较大，单机模拟的压测，轻易还压测不到扩容的水位。



总结

这节课我向你介绍了云原生基金会CNCF的重要成员：Kubernetes。**K8s是用于自动部署、扩展和管理容器化应用程序的开源系统。**云原生其实就是一套通用的云服务商解决方案。

然后我们一起体验了在本地PC上，通过Docker desktop搭建K8s。搭建完后，我还向你介绍了K8s的运行原理：K8s Master节点和Worker节点。其中，Master节点，负责我们整个K8s集群的运作状态；Worker节点则是具体运行我们容器的地方。

之后，我们就开始把“待办任务”Web服务，通过一个K8s的YAML文件来部署，并且暴露NodePort，让我们用浏览器访问。

为了展示K8s如何通过组件Component扩展能力，接着我们介绍了K8s中如何安装使用组件metrics：我们通过一个YAML文件将metrics组件安装到了K8s集群的kube-system命名空间中后，就可以监控到应用的运行指标metrics了。给K8s集群添加上监控指标metrics的能力，我们就可以通过autoscale命令，让应用根据metrics指标和水位来扩缩容了。

最后我们启动了一个BusyBox的Docker容器，模拟压测了我们的“待办任务”Web服务。

总的来说，这节课我们的最终目的就是在本地部署一套K8s集群，通过我们“待办任务”Web服务的K8s版本，让你了解K8s的工作原理。我们已经把下节课的准备工作做好了，下节课我们将在K8s的基础上部署Serverless，可以说，实现属于你自己的Serverless，你已经完成了一半。

作业

这节课是实战课，所以作业就是我们今天要练习的内容。请在你自己的电脑上安装K8s集群，部署我们的“待办任务”Web服务到自己的K8s集群，并从浏览器中访问到K8s集群提供的服务。

另外，你可以尝试一下，手动删除我们部署的MyApp Pod。

```
kubectl delete pod/你的pod名字
```

但你很快就会发现，这个Pod会被K8s重新拉起，而我们要清除部署的MyApp的所有内容其实也很简单，只需要告诉K8s删除我们的myapp.yaml文件创建的资源就可以了。

```
kubectl delete -f myapp.yaml
```

快来动手尝试一下吧，期待你也能分享下今天的成果以及感受。另外，如果今天的内容让你有所收获，也欢迎你把它分享给身边的朋友，邀请他加入学习。

参考资料

[1] https://gitforwindows.org/

[2] https://www.cncf.io/

[3] https://kubernetes.io/zh/

[4] https://github.com/cncf/landscape

[5] https://github.com/kubernetes-incubator/metrics-server/

你好，我是秦粤。上节课我们只是用Docker部署了index.js，如果我们将所有拆解出来的微服务都用Docker独立部署，我们就要同时管理多个Docker容器，也就是Docker集群。如果是更复杂一些的业务，可能需要同时管理几十甚至上百个微服务，显然我们手动维护Docker集群的效率就太低了。而容器即服务CaaS，恰好也需要集群的管理工具。我们也知道FaaS的底层就是CaaS，那CaaS又是如何管理这么多函数实例的呢？怎么做才能提升效率？

我想你应该听过Kubernetes，它也叫K8s（后面统一简称K8s），用于自动部署、扩展和管理容器化应用程序的开源系统，是Docker集群的管理工具。为了解决上述问题，其实我们就可以考虑使用它。K8s的好处就在于，它具备跨环境统一部署的能力。

这节课，我们就试着在本地环境中搭建K8s来管理我们的Docker集群。但正常情况下，这个场景需要几台机器才能完成，而通过Docker，我们还是可以用一台机器就可以在本地搭建一个低配版的K8s。

下节课，我们还会在今天内容的基础上，用K8s的CaaS方式实现一套Serverless环境。通过这两节课的内容，你就可以完整地搭建出属于自己的Serverless了。

话不多说，我们现在就开始，希望你能跟着我一起多动手。

PC上的K8s

那在开始之前，我们先得把安装问题解决了，这部分可能会有点小困难，所以我也给你详细讲下。

首先我们需要安装kubectl，它是K8s的命令行工具。

你需要在你的PC上安装K8s，如果你的操作系统是MacOS或者Windows，那么就比较简单了，桌面版的Docker已经自带了K8s；其它操作系统的同学需要安装minikube。

不过，要顺利启动桌面版Docker自带的K8s，你还得解决国内Docker镜像下载不了的问题，这里请你先下载第8课的代码。接着，请你跟着我的步骤进行操作：

1. 开通阿里云的容器镜像仓库；
2. 在阿里云的容器镜像服务里，找到镜像加速器，复制你的镜像加速器地址；
3. 打开桌面版Docker的控制台，找到Docker Engine。

```
{
  "registry-mirrors" : [
    "https://你的加速地址.mirror.aliyuncs.com"
  ],
  "debug" : true,
  "experimental" : true
}
```

4. 预下载K8s所需要的所有镜像，执行我目录下的docker-k8s-prefetch.sh，如果你是Windows操作系统，建议使用gitBash[1]；

```
chmode +x docker-k8s-prefetch.sh
./docker-k8s-prefetch.sh
```

5. 上面拉取完运行K8s所需的Docker镜像，你就可以在桌面版Docker的K8s项中，勾选启动K8s了。

现在，K8s就能顺利启动了，启动成功后，请你继续执行下面的命令。

查看安装好的K8s系统运行状况。

```
kubectl get all -n kube-system
```

查看K8s有哪些配置环境，对应~/.kube/config。

```
kubectl config get-contexts
```

查看当前K8s环境的整体情况。

```
kubectl get all
```

按照我的流程走，在大部分的机器上本地运行K8s都是没有问题的，如果你卡住了，请在留言区告知我，我帮你解决问题。

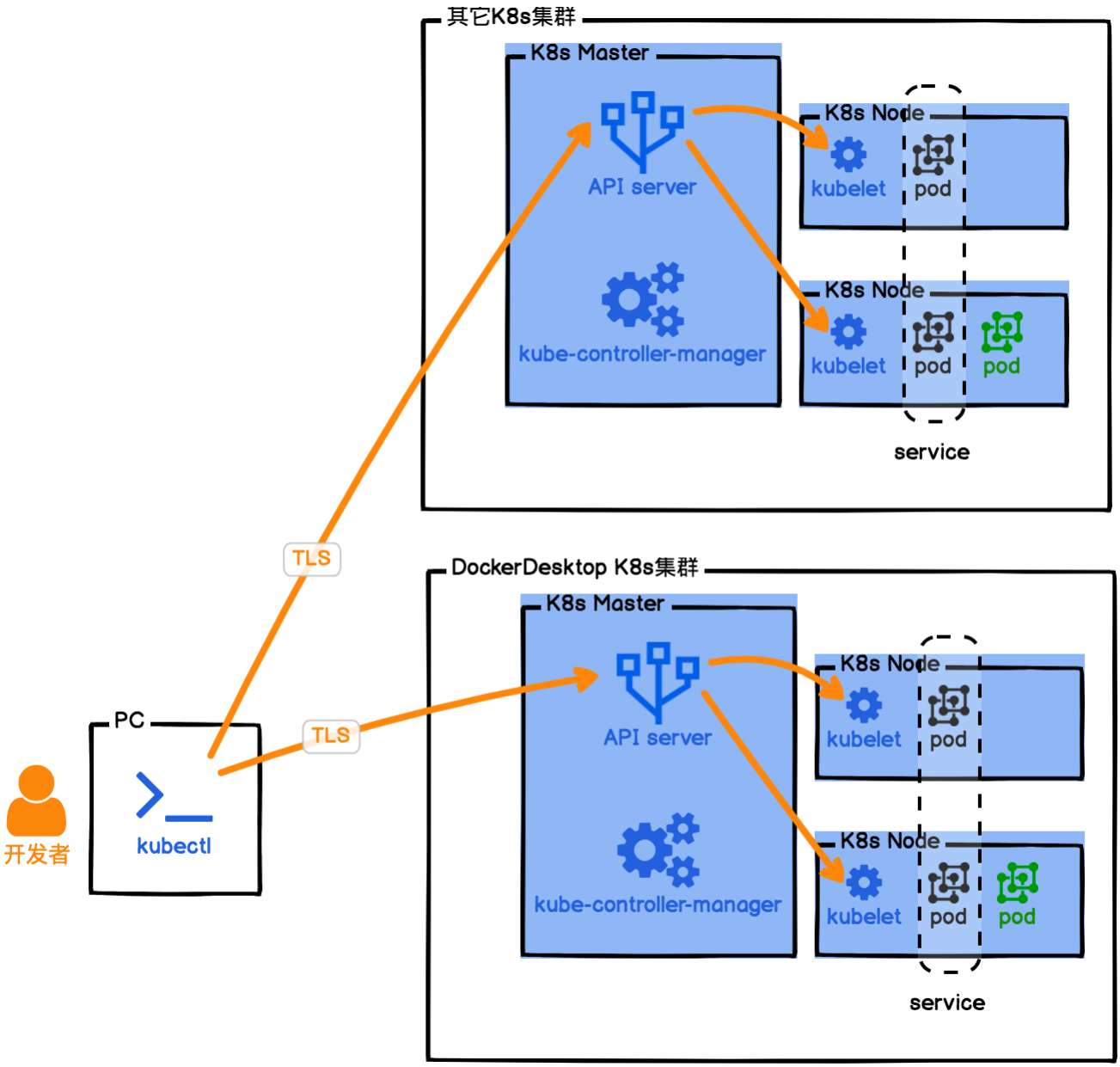
K8s介绍

安装完K8s之后，我们其实还是要简单地了解下K8s，这对于你后面应用它有重要的意义。

我想你应该知道，K8s是云原生Cloud Native[2] 的重要组成部分（目前，K8s的文档[3]已经全面中文化，建议你多翻阅），而云原生其实就是一套通用的云服务商解决方案。在我们的前几节课中，就有同学问：“如果我使用了某个运营商的Serverless，就和这个云服务商强制绑定了怎么办？我是不是就没办法使用其他运营商的服务了？”这就是服务商锁定vendor-lock，而云原生的诞生就是为了解决这个问题，通过云原生基金会CNCF(Cloud Native Computing Foundation)[4]，我们可以得到一整套解锁云服务商的开源解决方案。

那K8s作为云原生Cloud Native的重要组成部分之一，它的作用是什么呢？这里我先留一个悬念，通过理解K8s的原理，你就能清楚这个问题，并充分利用好K8s了。

我们先来看看K8s的原理图：



通过图示我们可以知道，PC本地安装kubectl是K8s的命令行操作工具，通过它，我们就可以控制K8s集群了。又因为kubectl是通过加密通信的，所以我们可以在一台电脑上同时控制多个K8s集群，不过需要指定当前操作的上下文context。这个也就是云原生的重要理念，我们的架构可以部署在多套云服务环境上。

在K8s集群中，Master节点很重要，它是我们整个集群的中枢。没错，Master节点就是Stateful的。Master节点由API Server、etcd、kube-controller-manager等几个核心成员组成，它只负责维持整个K8s集群的状态，为了保证职责单一，Master节点不会运行我们的容器实例。

Worker节点，也就是K8s Node节点，才是我们部署的容器真正运行的地方，但在K8s中，运行容器的最小单位是Pod。一个Pod具备一个集群IP且端口共享，Pod里可以运行一个或多个容器，但最佳的做法还是一个Pod只运行一个容器。这是因为一个Pod里面运行多个容器，容器会竞争Pod的资源，也会影响Pod的启动速度；而一个Pod里只运行一个容器，可以方便我们快速定位问题，监控指标也比较明确。

在K8s集群中，它会构建自己的私有网络，每个容器都有自己的集群IP，容器在集群内部可以互相访问，集群外却无法直接访问。因此我们如果要从外部访问K8s集群提供的服务，则需要通过K8s service将服务暴露出来才行。

案例：“待办任务”K8s版本

现在原理我是讲出来了，但可能还是有点不好理解，接下来我们就还是套进案例去看，依然是我们的“待办任务”Web服务，我们现在把它部署到K8s集群中运行一下，你可以切身体验。相信这样，你就非常清楚这其中的原理了。不过我们本地搭建的例子中，为了节省资源只有一个Master节点，所有的内容都部署在这个Master节点中。

还记得我们上节课构建的Docker镜像吗？我们就用它来部署本地的K8s集群。

我们通常在实际项目中会使用YAML文件来控制我们的应用部署。YAML你可以理解为，就是将我们在K8s部署的所有要做的事情，都写成一个文件，这样就避免了我们要记录大量的kubectl命令执行。不过，K8s也细心地帮我们准备了K8s对象和YAML文件互相转换的能力。这种能力可以让我们快速地将一个K8s集群中部署的结构导出YAML文件，然后再在另一个K8s集群中用这个YAML文件还原出同样的部署结构。

我们需要先确认一下，我们当前的操作是在正确的K8s集群上下文中。对应我们的例子里，也就是看当前选中的集群是不是docker-desktop。

```
kubectl config get-contexts
```

```
→  todolist-backend git:(lesson08) kubectl config get-contexts
CURRENT   NAME                 CLUSTER          AUTHINFO         NAMESPACE
*         docker-desktop       docker-desktop   docker-desktop
          docker-for-desktop   docker-desktop   docker-desktop
          minikube             minikube         minikube
```

如果不对，则需要执行切换集群：

```
kubectl config use-context docker-desktop
```

然后需要我们添加一下拉取镜像的证书服务：

```
kubectl create secret docker-registry regcred --docker-server=registry.cn-shanghai.aliyuncs.com --docker-username=你的容器镜像仓库用户名 --docker-password=你的容器镜像仓库密码
```

这里我需要解释一下，通常我们在镜像仓库中可以设置这个仓库：公开或者私有。如果是操作系统的镜像，设置为公开是完全没有问题的，所有人都可以下载我们的公开镜像；但如果是我们自己的应用镜像，还是需要设置成私有，下载私有镜像需要验证用户身份，也就是Docker Login的操作。因为我们应用的镜像仓库中，包含我们的最终运行代码，往往会有我们数据库的登录用户名和密码，或者我们云服务的ak/sk，这些重要信息如果泄露，很容易让我们的应用受到攻击。

当你添加完secret后，就可以通过下面的命令来查看secret服务了：

```
kubectl get secret regcred
```

另外我还需要再啰嗦一下，secret也不建议你用YAML文件设置，毕竟放着你用户名和密码的文件还是越少越好。

做完准备工作，对我们这次部署的项目而言就很简单了，只需要再执行一句话：

```
kubectl apply -f myapp.yaml
```

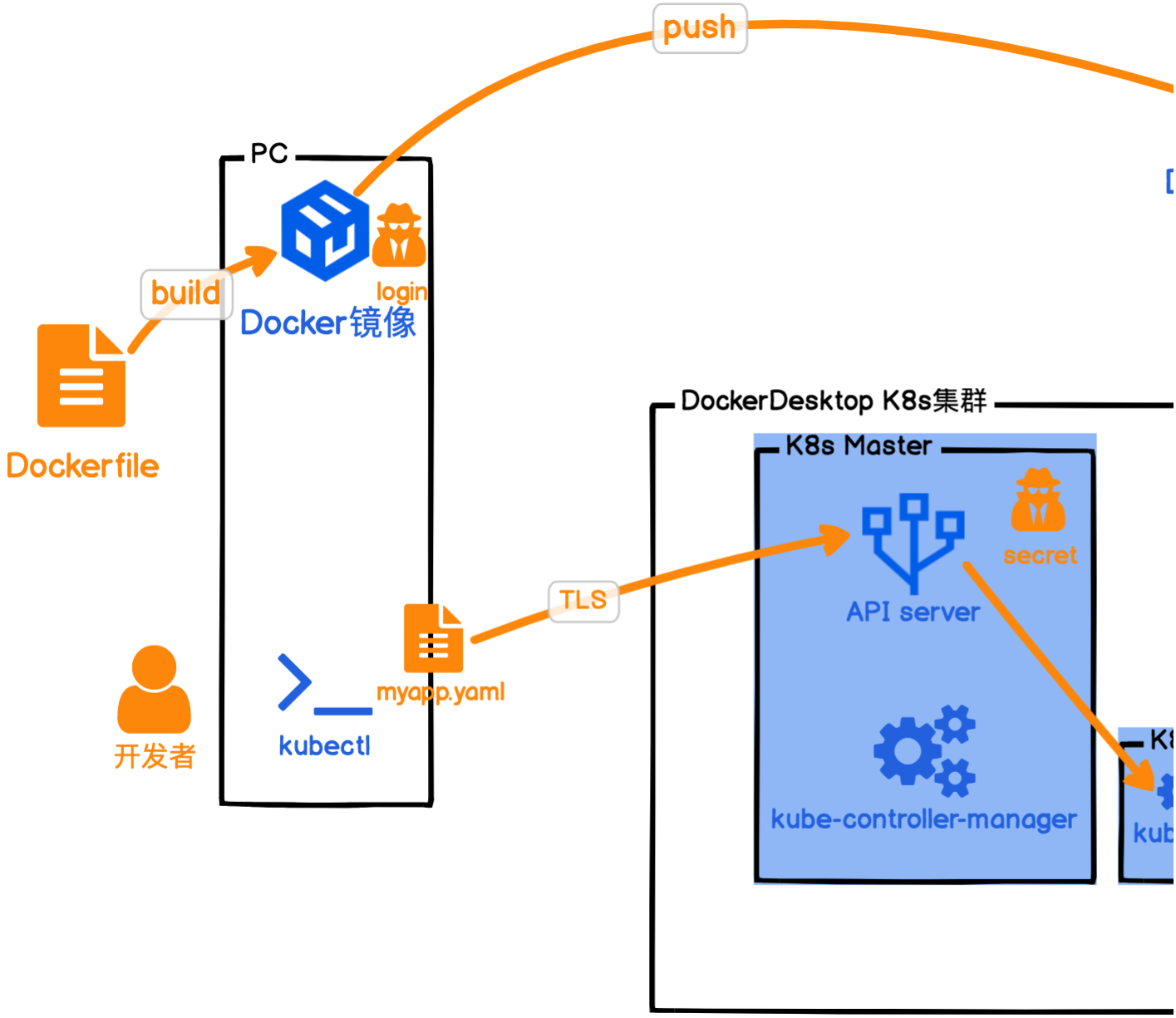
这句话的意思就是，告诉K8s集群，请按照我的部署文件myapp.yaml，部署我的应用。具体如下图所示：

```
➜  todolist-backend git:(lesson08) kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/myapp-759f79bd58-mjwwh   1/1     Running   2          21h

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          29h
service/myapp        NodePort    10.109.56.77   <none>        3001:30512/TCP   21h

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp   1/1     1            1           21h

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-759f79bd58   1         1         1       21h
```

通过获取容器的运行状态，对照上图我粗略地讲解一下我们的myapp.yaml文件吧。

- 首先我们指定要创建一个service/myapp，它选中打了"app:myapp"标签的Pod，集群内访问端口号3001，并且暴露service的端口号30512。
- 然后我们创建了部署服务deployment.apps/myapp，它负责保持我们的Pod数量恒久为1，并且给Pod打上"app:myapp"的标签，也就是负责我们的Pod持久化，一旦Pod挂了，部署服务就会重新拉起一个。
- 最后我们的容器服务，申明了自己的Docker镜像是什么，拉取镜像的secret，以及需要什么资源。

现在我们再回看K8s的原理图，不过这张是实现“待办任务”Web服务版本的：



首先我们可以看出，使用K8s仍然需要我们上节课的Docker发布流程：build、ship、run。不过现在有很多云服务商也会提供Docker镜像构建服务，你只需要上传你的Dockerfile，就会帮你构建镜像并且push到镜像仓库。云服务商提供的镜像构建服务的速度，比你本地构建要快很多倍。

而且相信你也发现了，K8s其实就是一套Docker容器实例的运行保障机制。我们自己Run一个Docker镜像，会有许多因素要考虑，例如安全性、网络隔离、日志、性能监控等等。这些K8s都帮我们考虑到了，它提供了一个Docker运行的最佳环境架构，而且还是开源的。

还有，既然我们本地都可以运行K8s的YAML文件，那么我们在云上是不是也能运行？你还记得前面我们留的悬念吧，现在就解决了。

通过K8s，我们要解开云服务商锁定vendor-lock就很简单了。我们只需要将云服务商提供的K8s环境添加到我们kubectl的配置文件中，就可以让我们的应用运行在云服务商的K8s环境中了。目前所有的较大的云服务商都已经加入CNCF，所以当你掌握K8s后，就可以根据云服务商的价格和自己喜好，自由地选择将你的K8s集群部署在CNCF成员的云服务商上，甚至你也可以在自己的机房搭建K8s环境，部署你的应用。

到这儿，我们就部署好了一个K8s集群，那之后我们该如何实时监控容器的运行状态呢？K8s既然能管理容器集群，控制容器运行实例的个数，那应该也能实时监测容器，帮我们解决扩缩容的问题吧？是的，其实上节课我们已经介绍到了容器扩缩容的原理，但并没有给你讲如何实现，那接下来我们就重点看看K8s如何实现实时监控和自动扩缩容。

K8s如何实现扩缩容？

首先，我们要知道的一点就是，K8s其实还向我们隐藏了一部分内容，就是它自身的服务状态。而我们不指定命名空间，默认的命名空间其实是default空间。要查看K8s集群系统的运行状态，我们可以通过指定namespace为kube-system来查看。K8s集群通过namespace隔离，一定程度上，隐藏了系统配置，这可以避免我们误操作。另外它也提供给我们一种逻辑隔离手段，将不同用途的服务和节点划分开来。

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          29h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          29h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          29h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/storage-provisioner                      1/1     Running   0          29h

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   29h

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns   2/2     2            2           29h

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4   2         2         2       29h
```

没错，K8s自己的服务也是运行在自己的集群中的，不过是通过命名空间，将自己做了隔离。这里需要你注意的是，这些服务我不建议你尝试去修改，因为它们涉及到了K8s集群的稳定性；但同时，K8s集群本身也具备扩展性：我们可以通过给K8s安装组件Component，扩展K8s的能力。接下来我先向你介绍K8s中的性能指标metrics组件[5]。

我的代码根目录下已经准备好了metric组件的YAML文件，你只需要执行安装就可以了：

```
kubectl apply -f metrics-components.yaml
```

安装完后，我们再看K8s的系统命名空间：

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          30h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          30h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          30h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/metrics-server-74657b4dc4-t979q          1/1     Running   0          100s
pod/storage-provisioner                      1/1     Running   0          29h

NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns         ClusterIP   10.96.0.10     <none>        53/UDP,53/TCP,9153/TCP   30h
service/metrics-server   ClusterIP   10.97.152.64   <none>        443/TCP                  100s

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns           2/2     2            2           30h
deployment.apps/metrics-server    1/1     1            1           100s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4         2         2         2       30h
replicaset.apps/metrics-server-74657b4dc4  1         1         1       100s
```

对比你就能发现，我们已经安装并启动了metrics-server。那么metrics组件有什么用呢？我们执行下面的命令看看：

```
kubectl top node
```

```
➜  todolist-backend git:(lesson08) kubectl top node
NAME             CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
docker-desktop   316m         7%     1249Mi          66%
```

安装metrics组件后，它就可以将我们应用的监控指标metrics显示出来了。没错，这里我们又可以用到上一讲的内容了。既然我们有了实时的监控指标，那么我们就可以依赖这个指标，来做我们的自动扩缩容了：

```
kubectl autoscale deployment myapp --cpu-percent=30 --min=1 --max=3
```

上面这句话的意思就是，添加一个自动扩缩容部署服务，cpu水位是30%，最小维持1个Pod，最大维持3个Pod。执行完后，我们就发现会多了一个部署服务。

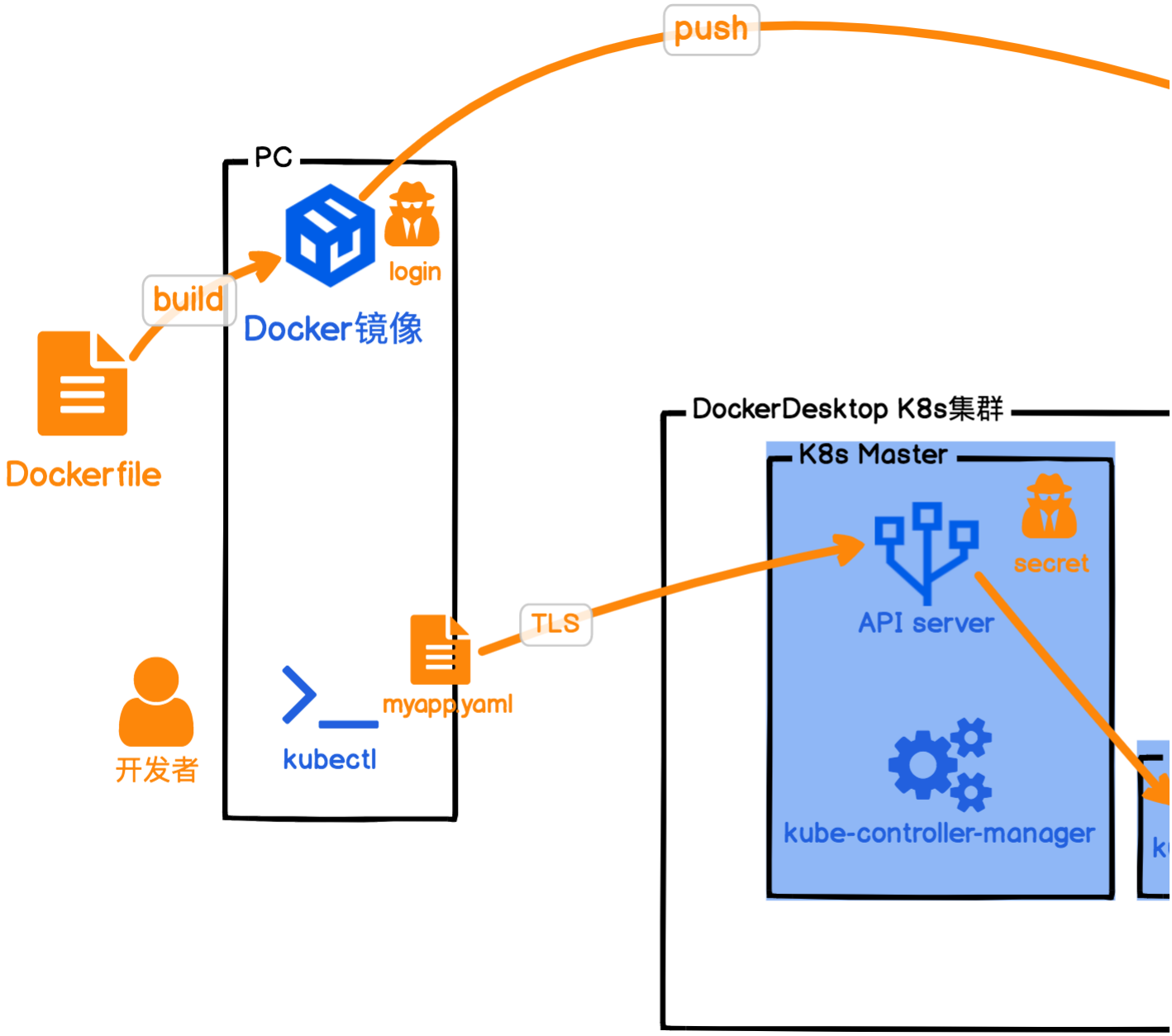
```
kubectl get hpa
```

```
→  todolist-backend git:(lesson08) kubectl get hpa
NAME     REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myapp    Deployment/myapp    0%/30%    1         3         1          22h
```

接下来，我们就可以模拟压测了：

```
kubectl run -i --tty load-generator --image=busybox /bin/sh
$ while true; do wget -q -O- http://10.1.0.16:3001/api/rule; done
```

这里我们用一个K8s的Pod，启动busybox镜像，执行死循环，压测我们的MyApp服务。不过我们目前用Node.js实现的应用可以扛住的流量比较大，单机模拟的压测，轻易还压测不到扩容的水位。



总结

这节课我向你介绍了云原生基金会CNCF的重要成员：Kubernetes。**K8s是用于自动部署、扩展和管理容器化应用程序的开源系统。**云原生其实就是一套通用的云服务商解决方案。

然后我们一起体验了在本地PC上，通过Docker desktop搭建K8s。搭建完后，我还向你介绍了K8s的运行原理：K8s Master节点和Worker节点。其中，Master节点，负责我们整个K8s集群的运作状态；Worker节点则是具体运行我们容器的地方。

之后，我们就开始把“待办任务”Web服务，通过一个K8s的YAML文件来部署，并且暴露NodePort，让我们用浏览器访问。

为了展示K8s如何通过组件Component扩展能力，接着我们介绍了K8s中如何安装使用组件metrics：我们通过一个YAML文件将metrics组件安装到了K8s集群的kube-system命名空间中后，就可以监控到应用的运行指标metrics了。给K8s集群添加上监控指标metrics的能力，我们就可以通过autoscale命令，让应用根据metrics指标和水位来扩缩容了。

最后我们启动了一个BusyBox的Docker容器，模拟压测了我们的“待办任务”Web服务。

总的来说，这节课我们的最终目的就是在本地部署一套K8s集群，通过我们“待办任务”Web服务的K8s版本，让你了解K8s的工作原理。我们已经把下节课的准备工作做好了，下节课我们将在K8s的基础上部署Serverless，可以说，实现属于你自己的Serverless，你已经完成了一半。

作业

这节课是实战课，所以作业就是我们今天要练习的内容。请在你自己的电脑上安装K8s集群，部署我们的“待办任务”Web服务到自己的K8s集群，并从浏览器中访问到K8s集群提供的服务。

另外，你可以尝试一下，手动删除我们部署的MyApp Pod。

```
kubectl delete pod/你的pod名字
```

但你很快就会发现，这个Pod会被K8s重新拉起，而我们要清除部署的MyApp的所有内容其实也很简单，只需要告诉K8s删除我们的myapp.yaml文件创建的资源就可以了。

```
kubectl delete -f myapp.yaml
```

快来动手尝试一下吧，期待你也能分享下今天的成果以及感受。另外，如果今天的内容让你有所收获，也欢迎你把它分享给身边的朋友，邀请他加入学习。

参考资料

[1] https://gitforwindows.org/

[2] https://www.cncf.io/

[3] https://kubernetes.io/zh/

[4] https://github.com/cncf/landscape

[5] https://github.com/kubernetes-incubator/metrics-server/

你好，我是秦粤。上节课我们只是用Docker部署了index.js，如果我们将所有拆解出来的微服务都用Docker独立部署，我们就要同时管理多个Docker容器，也就是Docker集群。如果是更复杂一些的业务，可能需要同时管理几十甚至上百个微服务，显然我们手动维护Docker集群的效率就太低了。而容器即服务CaaS，恰好也需要集群的管理工具。我们也知道FaaS的底层就是CaaS，那CaaS又是如何管理这么多函数实例的呢？怎么做才能提升效率？

我想你应该听过Kubernetes，它也叫K8s（后面统一简称K8s），用于自动部署、扩展和管理容器化应用程序的开源系统，是Docker集群的管理工具。为了解决上述问题，其实我们就可以考虑使用它。K8s的好处就在于，它具备跨环境统一部署的能力。

这节课，我们就试着在本地环境中搭建K8s来管理我们的Docker集群。但正常情况下，这个场景需要几台机器才能完成，而通过Docker，我们还是可以用一台机器就可以在本地搭建一个低配版的K8s。

下节课，我们还会在今天内容的基础上，用K8s的CaaS方式实现一套Serverless环境。通过这两节课的内容，你就可以完整地搭建出属于自己的Serverless了。

话不多说，我们现在就开始，希望你能跟着我一起多动手。

PC上的K8s

那在开始之前，我们先得把安装问题解决了，这部分可能会有点小困难，所以我也给你详细讲下。

首先我们需要安装kubectl，它是K8s的命令行工具。

你需要在你的PC上安装K8s，如果你的操作系统是MacOS或者Windows，那么就比较简单了，桌面版的Docker已经自带了K8s；其它操作系统的同学需要安装minikube。

不过，要顺利启动桌面版Docker自带的K8s，你还得解决国内Docker镜像下载不了的问题，这里请你先下载第8课的代码。接着，请你跟着我的步骤进行操作：

1. 开通阿里云的容器镜像仓库；
2. 在阿里云的容器镜像服务里，找到镜像加速器，复制你的镜像加速器地址；
3. 打开桌面版Docker的控制台，找到Docker Engine。

```
{
  "registry-mirrors" : [
    "https://你的加速地址.mirror.aliyuncs.com"
  ],
  "debug" : true,
  "experimental" : true
}
```

4. 预下载K8s所需要的所有镜像，执行我目录下的docker-k8s-prefetch.sh，如果你是Windows操作系统，建议使用gitBash[1]；

```
chmode +x docker-k8s-prefetch.sh
./docker-k8s-prefetch.sh
```

5. 上面拉取完运行K8s所需的Docker镜像，你就可以在桌面版Docker的K8s项中，勾选启动K8s了。

现在，K8s就能顺利启动了，启动成功后，请你继续执行下面的命令。

查看安装好的K8s系统运行状况。

```
kubectl get all -n kube-system
```

查看K8s有哪些配置环境，对应~/.kube/config。

```
kubectl config get-contexts
```

查看当前K8s环境的整体情况。

```
kubectl get all
```

按照我的流程走，在大部分的机器上本地运行K8s都是没有问题的，如果你卡住了，请在留言区告知我，我帮你解决问题。

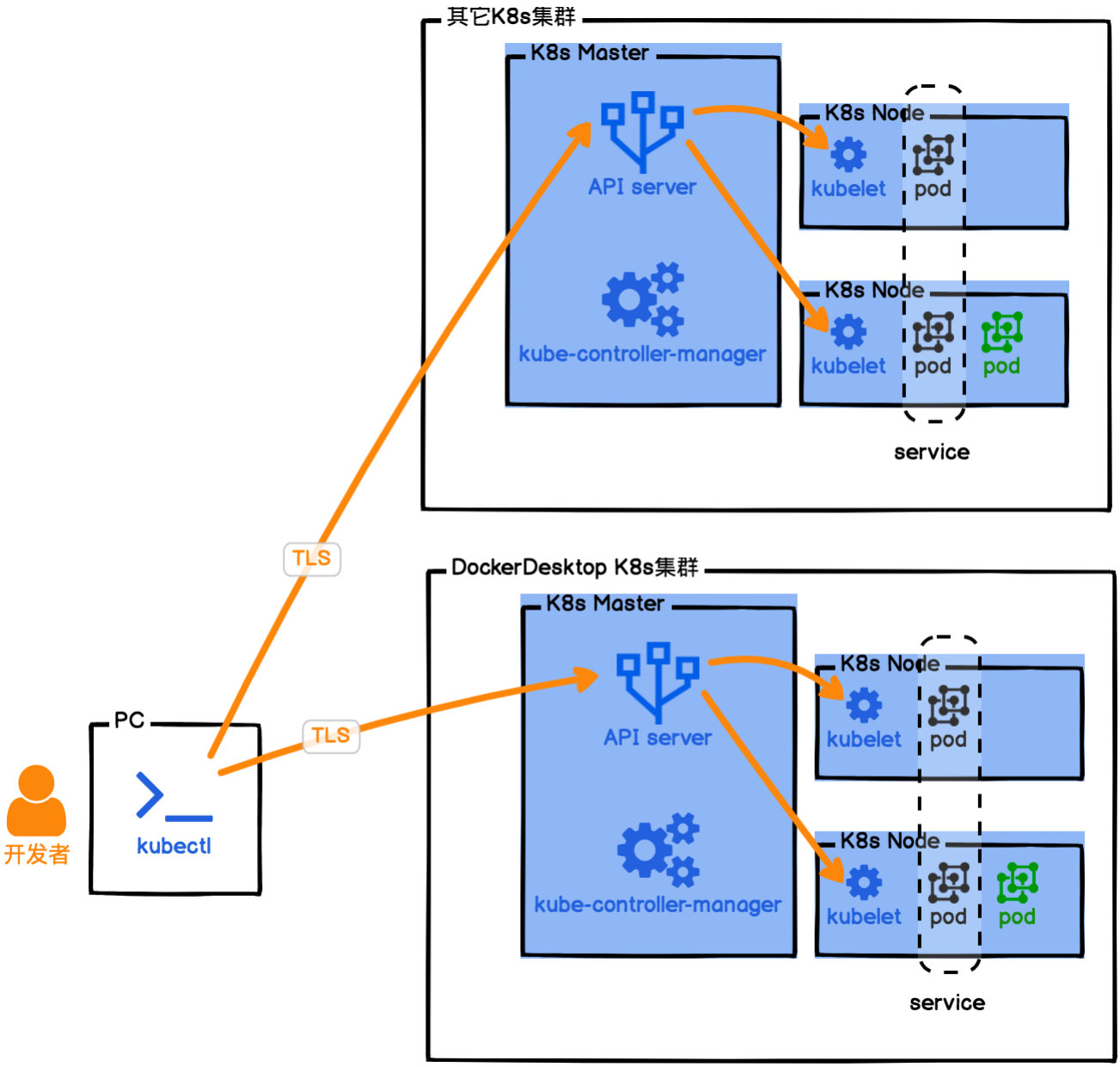
K8s介绍

安装完K8s之后，我们其实还是要简单地了解下K8s，这对于你后面应用它有重要的意义。

我想你应该知道，K8s是云原生Cloud Native[2] 的重要组成部分（目前，K8s的文档[3]已经全面中文化，建议你多翻阅），而云原生其实就是一套通用的云服务商解决方案。在我们的前几节课中，就有同学问：“如果我使用了某个运营商的Serverless，就和这个云服务商强制绑定了怎么办？我是不是就没办法使用其他运营商的服务了？”这就是服务商锁定vendor-lock，而云原生的诞生就是为了解决这个问题，通过云原生基金会CNCF(Cloud Native Computing Foundation)[4]，我们可以得到一整套解锁云服务商的开源解决方案。

那K8s作为云原生Cloud Native的重要组成部分之一，它的作用是什么呢？这里我先留一个悬念，通过理解K8s的原理，你就能清楚这个问题，并充分利用好K8s了。

我们先来看看K8s的原理图：



通过图示我们可以知道，PC本地安装kubectl是K8s的命令行操作工具，通过它，我们就可以控制K8s集群了。又因为kubectl是通过加密通信的，所以我们可以在一台电脑上同时控制多个K8s集群，不过需要指定当前操作的上下文context。这个也就是云原生的重要理念，我们的架构可以部署在多套云服务环境上。

在K8s集群中，Master节点很重要，它是我们整个集群的中枢。没错，Master节点就是Stateful的。Master节点由API Server、etcd、kube-controller-manager等几个核心成员组成，它只负责维持整个K8s集群的状态，为了保证职责单一，Master节点不会运行我们的容器实例。

Worker节点，也就是K8s Node节点，才是我们部署的容器真正运行的地方，但在K8s中，运行容器的最小单位是Pod。一个Pod具备一个集群IP且端口共享，Pod里可以运行一个或多个容器，但最佳的做法还是一个Pod只运行一个容器。这是因为一个Pod里面运行多个容器，容器会竞争Pod的资源，也会影响Pod的启动速度；而一个Pod里只运行一个容器，可以方便我们快速定位问题，监控指标也比较明确。

在K8s集群中，它会构建自己的私有网络，每个容器都有自己的集群IP，容器在集群内部可以互相访问，集群外却无法直接访问。因此我们如果要从外部访问K8s集群提供的服务，则需要通过K8s service将服务暴露出来才行。

案例：“待办任务”K8s版本

现在原理我是讲出来了，但可能还是有点不好理解，接下来我们就还是套进案例去看，依然是我们的“待办任务”Web服务，我们现在把它部署到K8s集群中运行一下，你可以切身体验。相信这样，你就非常清楚这其中的原理了。不过我们本地搭建的例子中，为了节省资源只有一个Master节点，所有的内容都部署在这个Master节点中。

还记得我们上节课构建的Docker镜像吗？我们就用它来部署本地的K8s集群。

我们通常在实际项目中会使用YAML文件来控制我们的应用部署。YAML你可以理解为，就是将我们在K8s部署的所有要做的事情，都写成一个文件，这样就避免了我们要记录大量的kubectl命令执行。不过，K8s也细心地帮我们准备了K8s对象和YAML文件互相转换的能力。这种能力可以让我们快速地将一个K8s集群中部署的结构导出YAML文件，然后再在另一个K8s集群中用这个YAML文件还原出同样的部署结构。

我们需要先确认一下，我们当前的操作是在正确的K8s集群上下文中。对应我们的例子里，也就是看当前选中的集群是不是docker-desktop。

```
kubectl config get-contexts
```

```
→  todolist-backend git:(lesson08) kubectl config get-contexts
CURRENT   NAME                 CLUSTER          AUTHINFO         NAMESPACE
*         docker-desktop       docker-desktop   docker-desktop
          docker-for-desktop   docker-desktop   docker-desktop
          minikube             minikube         minikube
```

如果不对，则需要执行切换集群：

```
kubectl config use-context docker-desktop
```

然后需要我们添加一下拉取镜像的证书服务：

```
kubectl create secret docker-registry regcred --docker-server=registry.cn-shanghai.aliyuncs.com --docker-username=你的容器镜像仓库用户名 --docker-password=你的容器镜像仓库密码
```

这里我需要解释一下，通常我们在镜像仓库中可以设置这个仓库：公开或者私有。如果是操作系统的镜像，设置为公开是完全没有问题的，所有人都可以下载我们的公开镜像；但如果是我们自己的应用镜像，还是需要设置成私有，下载私有镜像需要验证用户身份，也就是Docker Login的操作。因为我们应用的镜像仓库中，包含我们的最终运行代码，往往会有我们数据库的登录用户名和密码，或者我们云服务的ak/sk，这些重要信息如果泄露，很容易让我们的应用受到攻击。

当你添加完secret后，就可以通过下面的命令来查看secret服务了：

```
kubectl get secret regcred
```

另外我还需要再啰嗦一下，secret也不建议你用YAML文件设置，毕竟放着你用户名和密码的文件还是越少越好。

做完准备工作，对我们这次部署的项目而言就很简单了，只需要再执行一句话：

```
kubectl apply -f myapp.yaml
```

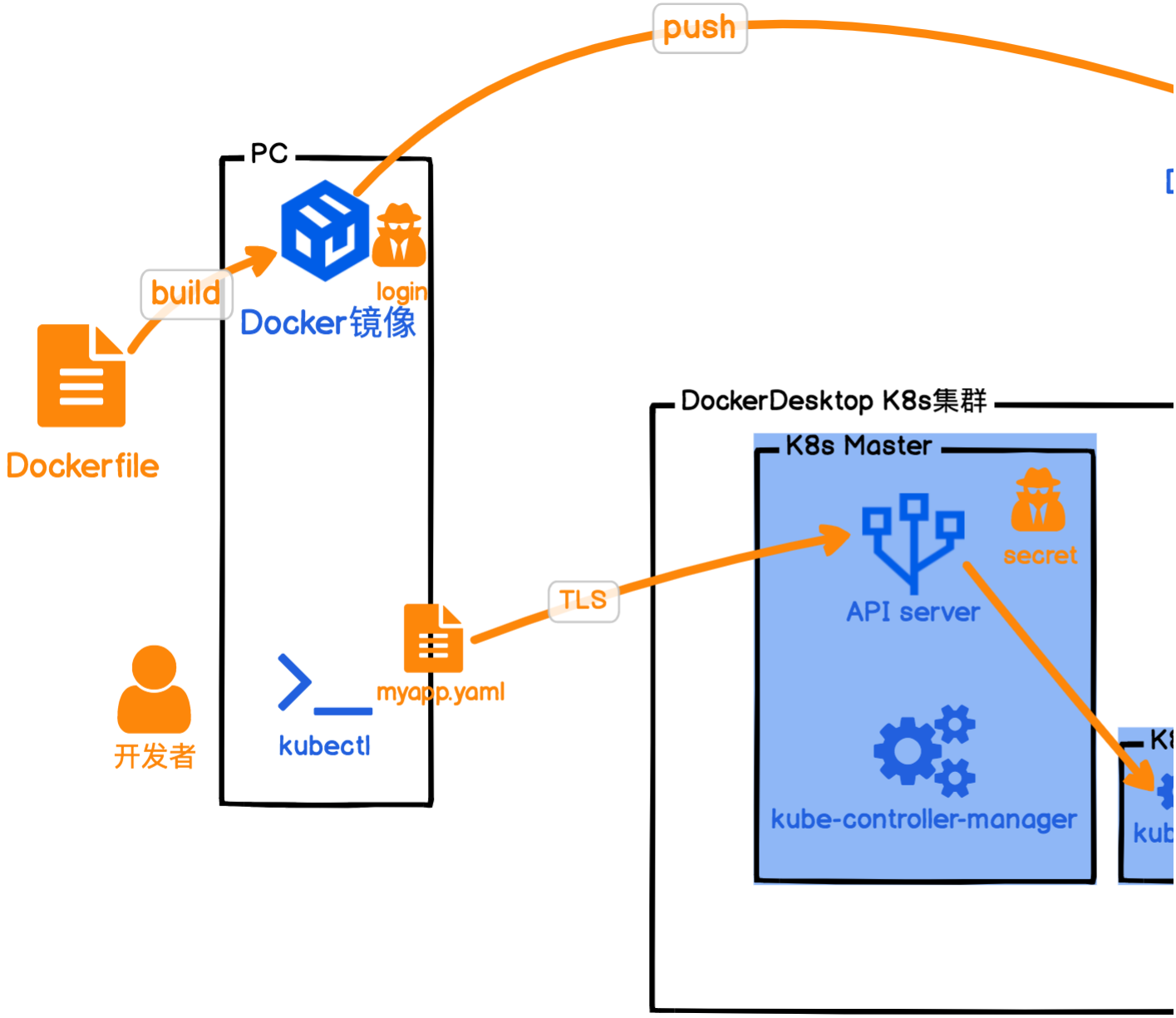
这句话的意思就是，告诉K8s集群，请按照我的部署文件myapp.yaml，部署我的应用。具体如下图所示：

```
➜  todolist-backend git:(lesson08) kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/myapp-759f79bd58-mjwwh   1/1     Running   2          21h

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          29h
service/myapp        NodePort    10.109.56.77   <none>        3001:30512/TCP   21h

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp   1/1     1            1           21h

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-759f79bd58   1         1         1       21h
```

通过获取容器的运行状态，对照上图我粗略地讲解一下我们的myapp.yaml文件吧。

- 首先我们指定要创建一个service/myapp，它选中打了"app:myapp"标签的Pod，集群内访问端口号3001，并且暴露service的端口号30512。
- 然后我们创建了部署服务deployment.apps/myapp，它负责保持我们的Pod数量恒久为1，并且给Pod打上"app:myapp"的标签，也就是负责我们的Pod持久化，一旦Pod挂了，部署服务就会重新拉起一个。
- 最后我们的容器服务，申明了自己的Docker镜像是什么，拉取镜像的secret，以及需要什么资源。

现在我们再回看K8s的原理图，不过这张是实现“待办任务”Web服务版本的：



首先我们可以看出，使用K8s仍然需要我们上节课的Docker发布流程：build、ship、run。不过现在有很多云服务商也会提供Docker镜像构建服务，你只需要上传你的Dockerfile，就会帮你构建镜像并且push到镜像仓库。云服务商提供的镜像构建服务的速度，比你本地构建要快很多倍。

而且相信你也发现了，K8s其实就是一套Docker容器实例的运行保障机制。我们自己Run一个Docker镜像，会有许多因素要考虑，例如安全性、网络隔离、日志、性能监控等等。这些K8s都帮我们考虑到了，它提供了一个Docker运行的最佳环境架构，而且还是开源的。

还有，既然我们本地都可以运行K8s的YAML文件，那么我们在云上是不是也能运行？你还记得前面我们留的悬念吧，现在就解决了。

通过K8s，我们要解开云服务商锁定vendor-lock就很简单了。我们只需要将云服务商提供的K8s环境添加到我们kubectl的配置文件中，就可以让我们的应用运行在云服务商的K8s环境中了。目前所有的较大的云服务商都已经加入CNCF，所以当你掌握K8s后，就可以根据云服务商的价格和自己喜好，自由地选择将你的K8s集群部署在CNCF成员的云服务商上，甚至你也可以在自己的机房搭建K8s环境，部署你的应用。

到这儿，我们就部署好了一个K8s集群，那之后我们该如何实时监控容器的运行状态呢？K8s既然能管理容器集群，控制容器运行实例的个数，那应该也能实时监测容器，帮我们解决扩缩容的问题吧？是的，其实上节课我们已经介绍到了容器扩缩容的原理，但并没有给你讲如何实现，那接下来我们就重点看看K8s如何实现实时监控和自动扩缩容。

K8s如何实现扩缩容？

首先，我们要知道的一点就是，K8s其实还向我们隐藏了一部分内容，就是它自身的服务状态。而我们不指定命名空间，默认的命名空间其实是default空间。要查看K8s集群系统的运行状态，我们可以通过指定namespace为kube-system来查看。K8s集群通过namespace隔离，一定程度上，隐藏了系统配置，这可以避免我们误操作。另外它也提供给我们一种逻辑隔离手段，将不同用途的服务和节点划分开来。

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          29h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          29h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          29h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/storage-provisioner                      1/1     Running   0          29h

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   29h

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns   2/2     2            2           29h

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4   2         2         2       29h
```

没错，K8s自己的服务也是运行在自己的集群中的，不过是通过命名空间，将自己做了隔离。这里需要你注意的是，这些服务我不建议你尝试去修改，因为它们涉及到了K8s集群的稳定性；但同时，K8s集群本身也具备扩展性：我们可以通过给K8s安装组件Component，扩展K8s的能力。接下来我先向你介绍K8s中的性能指标metrics组件[5]。

我的代码根目录下已经准备好了metric组件的YAML文件，你只需要执行安装就可以了：

```
kubectl apply -f metrics-components.yaml
```

安装完后，我们再看K8s的系统命名空间：

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          30h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          30h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          30h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/metrics-server-74657b4dc4-t979q          1/1     Running   0          100s
pod/storage-provisioner                      1/1     Running   0          29h

NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns         ClusterIP   10.96.0.10     <none>        53/UDP,53/TCP,9153/TCP   30h
service/metrics-server   ClusterIP   10.97.152.64   <none>        443/TCP                  100s

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns           2/2     2            2           30h
deployment.apps/metrics-server    1/1     1            1           100s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4         2         2         2       30h
replicaset.apps/metrics-server-74657b4dc4  1         1         1       100s
```

对比你就能发现，我们已经安装并启动了metrics-server。那么metrics组件有什么用呢？我们执行下面的命令看看：

```
kubectl top node
```

```
➜  todolist-backend git:(lesson08) kubectl top node
NAME             CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
docker-desktop   316m         7%     1249Mi          66%
```

安装metrics组件后，它就可以将我们应用的监控指标metrics显示出来了。没错，这里我们又可以用到上一讲的内容了。既然我们有了实时的监控指标，那么我们就可以依赖这个指标，来做我们的自动扩缩容了：

```
kubectl autoscale deployment myapp --cpu-percent=30 --min=1 --max=3
```

上面这句话的意思就是，添加一个自动扩缩容部署服务，cpu水位是30%，最小维持1个Pod，最大维持3个Pod。执行完后，我们就发现会多了一个部署服务。

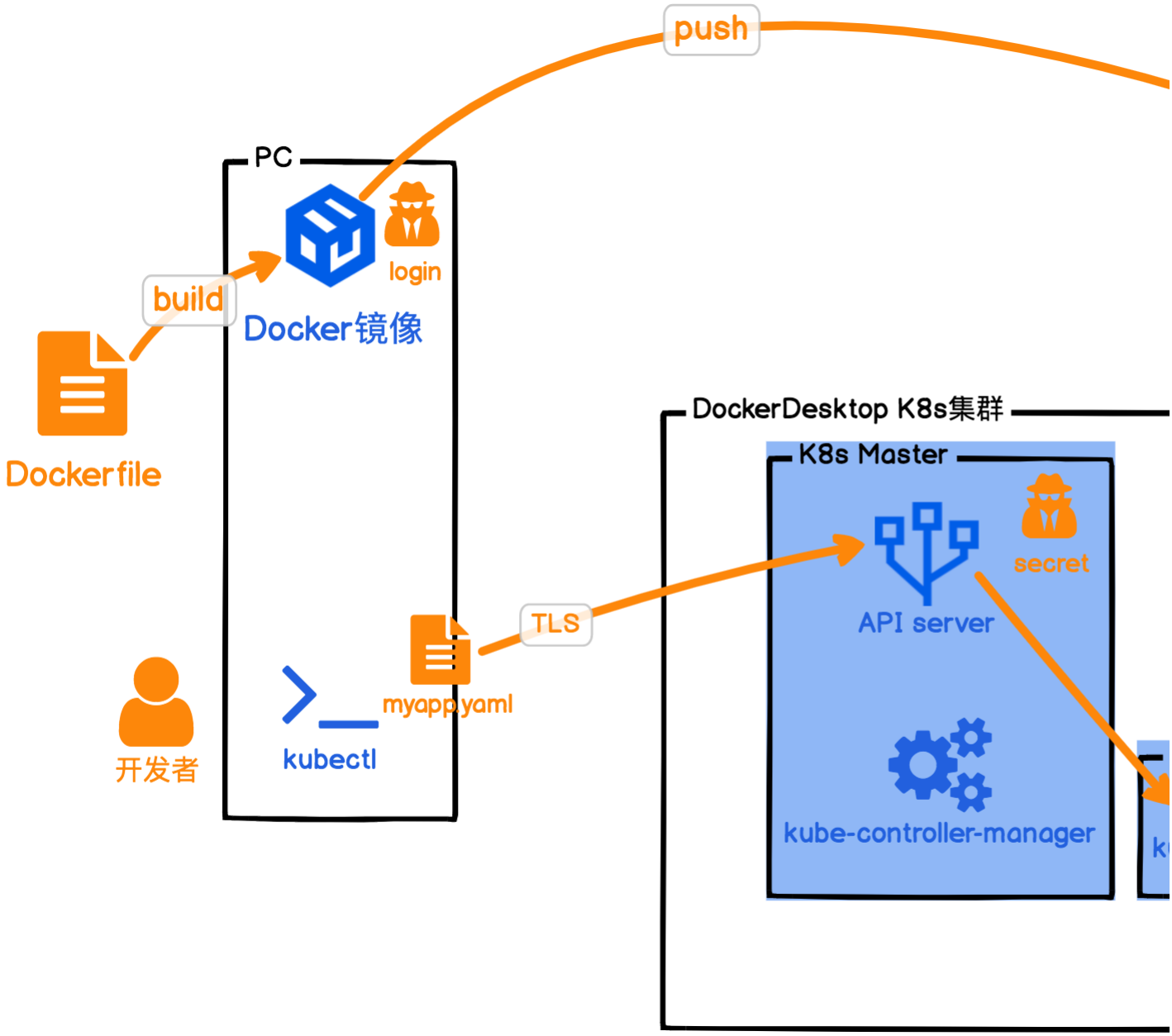
```
kubectl get hpa
```

```
→  todolist-backend git:(lesson08) kubectl get hpa
NAME     REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myapp    Deployment/myapp    0%/30%    1         3         1          22h
```

接下来，我们就可以模拟压测了：

```
kubectl run -i --tty load-generator --image=busybox /bin/sh
$ while true; do wget -q -O- http://10.1.0.16:3001/api/rule; done
```

这里我们用一个K8s的Pod，启动busybox镜像，执行死循环，压测我们的MyApp服务。不过我们目前用Node.js实现的应用可以扛住的流量比较大，单机模拟的压测，轻易还压测不到扩容的水位。



总结

这节课我向你介绍了云原生基金会CNCF的重要成员：Kubernetes。**K8s是用于自动部署、扩展和管理容器化应用程序的开源系统。**云原生其实就是一套通用的云服务商解决方案。

然后我们一起体验了在本地PC上，通过Docker desktop搭建K8s。搭建完后，我还向你介绍了K8s的运行原理：K8s Master节点和Worker节点。其中，Master节点，负责我们整个K8s集群的运作状态；Worker节点则是具体运行我们容器的地方。

之后，我们就开始把“待办任务”Web服务，通过一个K8s的YAML文件来部署，并且暴露NodePort，让我们用浏览器访问。

为了展示K8s如何通过组件Component扩展能力，接着我们介绍了K8s中如何安装使用组件metrics：我们通过一个YAML文件将metrics组件安装到了K8s集群的kube-system命名空间中后，就可以监控到应用的运行指标metrics了。给K8s集群添加上监控指标metrics的能力，我们就可以通过autoscale命令，让应用根据metrics指标和水位来扩缩容了。

最后我们启动了一个BusyBox的Docker容器，模拟压测了我们的“待办任务”Web服务。

总的来说，这节课我们的最终目的就是在本地部署一套K8s集群，通过我们“待办任务”Web服务的K8s版本，让你了解K8s的工作原理。我们已经把下节课的准备工作做好了，下节课我们将在K8s的基础上部署Serverless，可以说，实现属于你自己的Serverless，你已经完成了一半。

作业

这节课是实战课，所以作业就是我们今天要练习的内容。请在你自己的电脑上安装K8s集群，部署我们的“待办任务”Web服务到自己的K8s集群，并从浏览器中访问到K8s集群提供的服务。

另外，你可以尝试一下，手动删除我们部署的MyApp Pod。

```
kubectl delete pod/你的pod名字
```

但你很快就会发现，这个Pod会被K8s重新拉起，而我们要清除部署的MyApp的所有内容其实也很简单，只需要告诉K8s删除我们的myapp.yaml文件创建的资源就可以了。

```
kubectl delete -f myapp.yaml
```

快来动手尝试一下吧，期待你也能分享下今天的成果以及感受。另外，如果今天的内容让你有所收获，也欢迎你把它分享给身边的朋友，邀请他加入学习。

参考资料

[1] https://gitforwindows.org/

[2] https://www.cncf.io/

[3] https://kubernetes.io/zh/

[4] https://github.com/cncf/landscape

[5] https://github.com/kubernetes-incubator/metrics-server/

你好，我是秦粤。上节课我们只是用Docker部署了index.js，如果我们将所有拆解出来的微服务都用Docker独立部署，我们就要同时管理多个Docker容器，也就是Docker集群。如果是更复杂一些的业务，可能需要同时管理几十甚至上百个微服务，显然我们手动维护Docker集群的效率就太低了。而容器即服务CaaS，恰好也需要集群的管理工具。我们也知道FaaS的底层就是CaaS，那CaaS又是如何管理这么多函数实例的呢？怎么做才能提升效率？

我想你应该听过Kubernetes，它也叫K8s（后面统一简称K8s），用于自动部署、扩展和管理容器化应用程序的开源系统，是Docker集群的管理工具。为了解决上述问题，其实我们就可以考虑使用它。K8s的好处就在于，它具备跨环境统一部署的能力。

这节课，我们就试着在本地环境中搭建K8s来管理我们的Docker集群。但正常情况下，这个场景需要几台机器才能完成，而通过Docker，我们还是可以用一台机器就可以在本地搭建一个低配版的K8s。

下节课，我们还会在今天内容的基础上，用K8s的CaaS方式实现一套Serverless环境。通过这两节课的内容，你就可以完整地搭建出属于自己的Serverless了。

话不多说，我们现在就开始，希望你能跟着我一起多动手。

PC上的K8s

那在开始之前，我们先得把安装问题解决了，这部分可能会有点小困难，所以我也给你详细讲下。

首先我们需要安装kubectl，它是K8s的命令行工具。

你需要在你的PC上安装K8s，如果你的操作系统是MacOS或者Windows，那么就比较简单了，桌面版的Docker已经自带了K8s；其它操作系统的同学需要安装minikube。

不过，要顺利启动桌面版Docker自带的K8s，你还得解决国内Docker镜像下载不了的问题，这里请你先下载第8课的代码。接着，请你跟着我的步骤进行操作：

1. 开通阿里云的容器镜像仓库；
2. 在阿里云的容器镜像服务里，找到镜像加速器，复制你的镜像加速器地址；
3. 打开桌面版Docker的控制台，找到Docker Engine。

```
{
  "registry-mirrors" : [
    "https://你的加速地址.mirror.aliyuncs.com"
  ],
  "debug" : true,
  "experimental" : true
}
```

4. 预下载K8s所需要的所有镜像，执行我目录下的docker-k8s-prefetch.sh，如果你是Windows操作系统，建议使用gitBash[1]；

```
chmode +x docker-k8s-prefetch.sh
./docker-k8s-prefetch.sh
```

5. 上面拉取完运行K8s所需的Docker镜像，你就可以在桌面版Docker的K8s项中，勾选启动K8s了。

现在，K8s就能顺利启动了，启动成功后，请你继续执行下面的命令。

查看安装好的K8s系统运行状况。

```
kubectl get all -n kube-system
```

查看K8s有哪些配置环境，对应~/.kube/config。

```
kubectl config get-contexts
```

查看当前K8s环境的整体情况。

```
kubectl get all
```

按照我的流程走，在大部分的机器上本地运行K8s都是没有问题的，如果你卡住了，请在留言区告知我，我帮你解决问题。

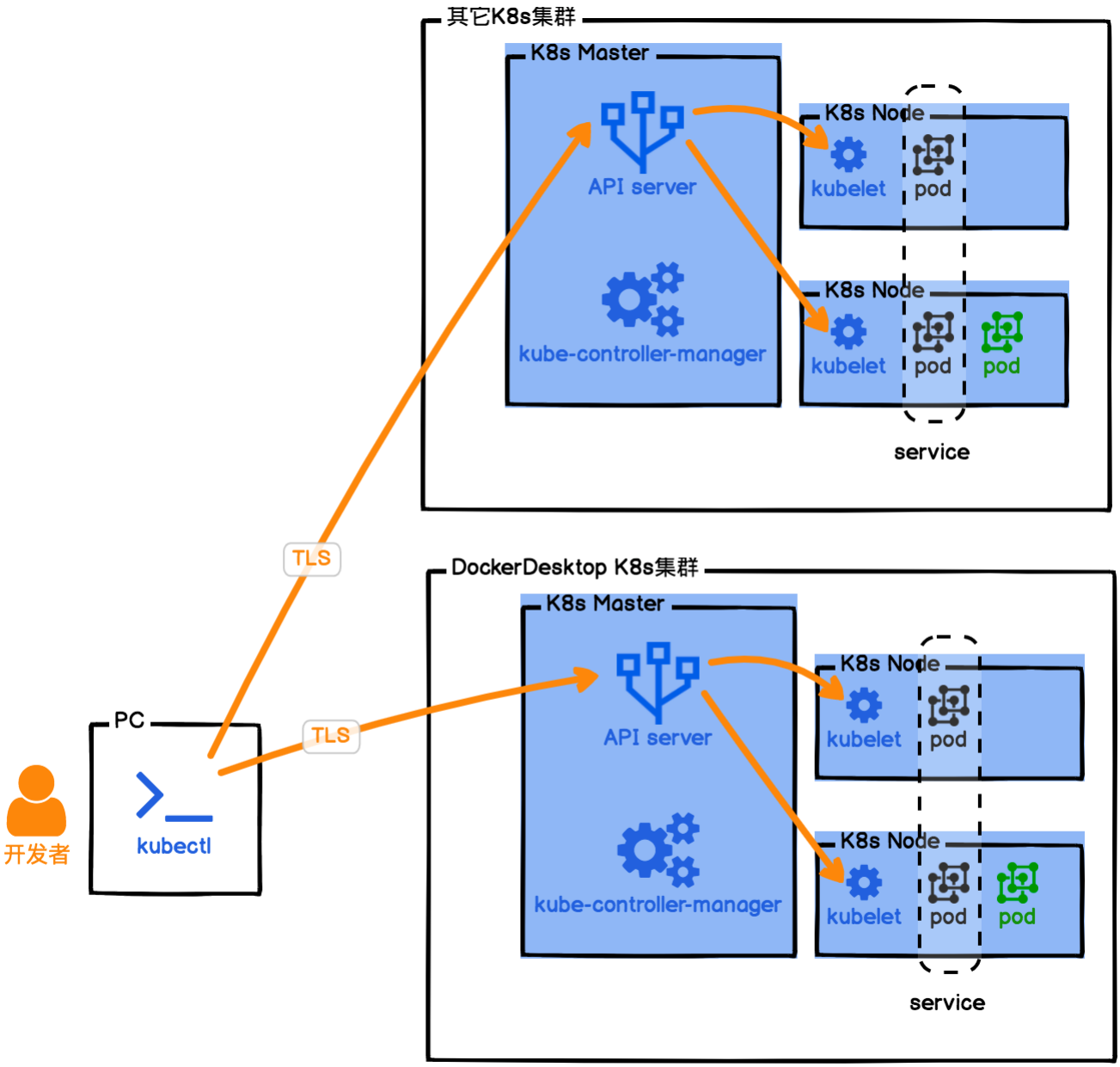
K8s介绍

安装完K8s之后，我们其实还是要简单地了解下K8s，这对于你后面应用它有重要的意义。

我想你应该知道，K8s是云原生Cloud Native[2] 的重要组成部分（目前，K8s的文档[3]已经全面中文化，建议你多翻阅），而云原生其实就是一套通用的云服务商解决方案。在我们的前几节课中，就有同学问：“如果我使用了某个运营商的Serverless，就和这个云服务商强制绑定了怎么办？我是不是就没办法使用其他运营商的服务了？”这就是服务商锁定vendor-lock，而云原生的诞生就是为了解决这个问题，通过云原生基金会CNCF(Cloud Native Computing Foundation)[4]，我们可以得到一整套解锁云服务商的开源解决方案。

那K8s作为云原生Cloud Native的重要组成部分之一，它的作用是什么呢？这里我先留一个悬念，通过理解K8s的原理，你就能清楚这个问题，并充分利用好K8s了。

我们先来看看K8s的原理图：



通过图示我们可以知道，PC本地安装kubectl是K8s的命令行操作工具，通过它，我们就可以控制K8s集群了。又因为kubectl是通过加密通信的，所以我们可以在一台电脑上同时控制多个K8s集群，不过需要指定当前操作的上下文context。这个也就是云原生的重要理念，我们的架构可以部署在多套云服务环境上。

在K8s集群中，Master节点很重要，它是我们整个集群的中枢。没错，Master节点就是Stateful的。Master节点由API Server、etcd、kube-controller-manager等几个核心成员组成，它只负责维持整个K8s集群的状态，为了保证职责单一，Master节点不会运行我们的容器实例。

Worker节点，也就是K8s Node节点，才是我们部署的容器真正运行的地方，但在K8s中，运行容器的最小单位是Pod。一个Pod具备一个集群IP且端口共享，Pod里可以运行一个或多个容器，但最佳的做法还是一个Pod只运行一个容器。这是因为一个Pod里面运行多个容器，容器会竞争Pod的资源，也会影响Pod的启动速度；而一个Pod里只运行一个容器，可以方便我们快速定位问题，监控指标也比较明确。

在K8s集群中，它会构建自己的私有网络，每个容器都有自己的集群IP，容器在集群内部可以互相访问，集群外却无法直接访问。因此我们如果要从外部访问K8s集群提供的服务，则需要通过K8s service将服务暴露出来才行。

案例：“待办任务”K8s版本

现在原理我是讲出来了，但可能还是有点不好理解，接下来我们就还是套进案例去看，依然是我们的“待办任务”Web服务，我们现在把它部署到K8s集群中运行一下，你可以切身体验。相信这样，你就非常清楚这其中的原理了。不过我们本地搭建的例子中，为了节省资源只有一个Master节点，所有的内容都部署在这个Master节点中。

还记得我们上节课构建的Docker镜像吗？我们就用它来部署本地的K8s集群。

我们通常在实际项目中会使用YAML文件来控制我们的应用部署。YAML你可以理解为，就是将我们在K8s部署的所有要做的事情，都写成一个文件，这样就避免了我们要记录大量的kubectl命令执行。不过，K8s也细心地帮我们准备了K8s对象和YAML文件互相转换的能力。这种能力可以让我们快速地将一个K8s集群中部署的结构导出YAML文件，然后再在另一个K8s集群中用这个YAML文件还原出同样的部署结构。

我们需要先确认一下，我们当前的操作是在正确的K8s集群上下文中。对应我们的例子里，也就是看当前选中的集群是不是docker-desktop。

```
kubectl config get-contexts
```

```
→  todolist-backend git:(lesson08) kubectl config get-contexts
CURRENT   NAME                 CLUSTER          AUTHINFO         NAMESPACE
*         docker-desktop       docker-desktop   docker-desktop
          docker-for-desktop   docker-desktop   docker-desktop
          minikube             minikube         minikube
```

如果不对，则需要执行切换集群：

```
kubectl config use-context docker-desktop
```

然后需要我们添加一下拉取镜像的证书服务：

```
kubectl create secret docker-registry regcred --docker-server=registry.cn-shanghai.aliyuncs.com --docker-username=你的容器镜像仓库用户名 --docker-password=你的容器镜像仓库密码
```

这里我需要解释一下，通常我们在镜像仓库中可以设置这个仓库：公开或者私有。如果是操作系统的镜像，设置为公开是完全没有问题的，所有人都可以下载我们的公开镜像；但如果是我们自己的应用镜像，还是需要设置成私有，下载私有镜像需要验证用户身份，也就是Docker Login的操作。因为我们应用的镜像仓库中，包含我们的最终运行代码，往往会有我们数据库的登录用户名和密码，或者我们云服务的ak/sk，这些重要信息如果泄露，很容易让我们的应用受到攻击。

当你添加完secret后，就可以通过下面的命令来查看secret服务了：

```
kubectl get secret regcred
```

另外我还需要再啰嗦一下，secret也不建议你用YAML文件设置，毕竟放着你用户名和密码的文件还是越少越好。

做完准备工作，对我们这次部署的项目而言就很简单了，只需要再执行一句话：

```
kubectl apply -f myapp.yaml
```

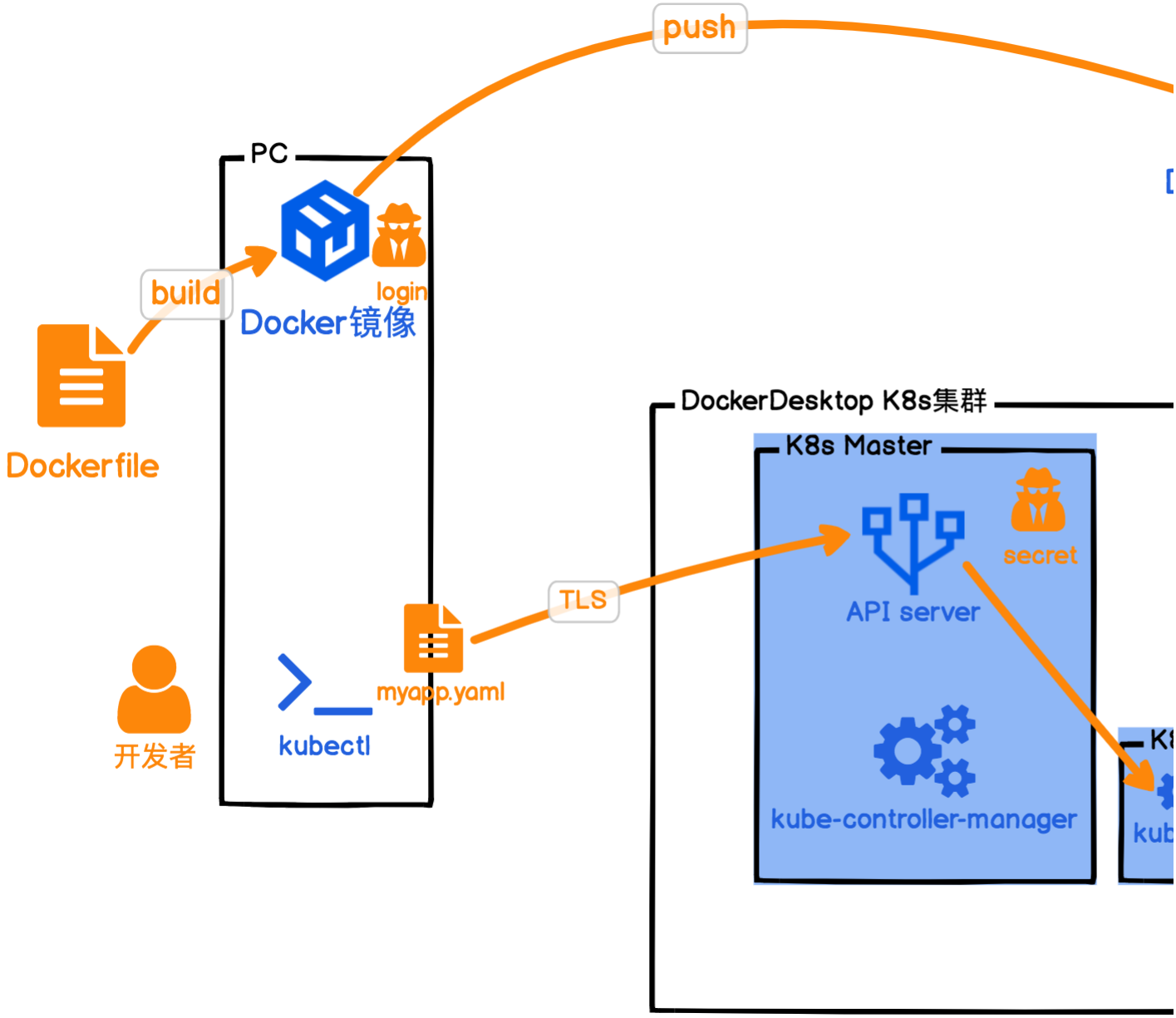
这句话的意思就是，告诉K8s集群，请按照我的部署文件myapp.yaml，部署我的应用。具体如下图所示：

```
➜  todolist-backend git:(lesson08) kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/myapp-759f79bd58-mjwwh   1/1     Running   2          21h

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          29h
service/myapp        NodePort    10.109.56.77   <none>        3001:30512/TCP   21h

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp   1/1     1            1           21h

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-759f79bd58   1         1         1       21h
```

通过获取容器的运行状态，对照上图我粗略地讲解一下我们的myapp.yaml文件吧。

- 首先我们指定要创建一个service/myapp，它选中打了"app:myapp"标签的Pod，集群内访问端口号3001，并且暴露service的端口号30512。
- 然后我们创建了部署服务deployment.apps/myapp，它负责保持我们的Pod数量恒久为1，并且给Pod打上"app:myapp"的标签，也就是负责我们的Pod持久化，一旦Pod挂了，部署服务就会重新拉起一个。
- 最后我们的容器服务，申明了自己的Docker镜像是什么，拉取镜像的secret，以及需要什么资源。

现在我们再回看K8s的原理图，不过这张是实现“待办任务”Web服务版本的：



首先我们可以看出，使用K8s仍然需要我们上节课的Docker发布流程：build、ship、run。不过现在有很多云服务商也会提供Docker镜像构建服务，你只需要上传你的Dockerfile，就会帮你构建镜像并且push到镜像仓库。云服务商提供的镜像构建服务的速度，比你本地构建要快很多倍。

而且相信你也发现了，K8s其实就是一套Docker容器实例的运行保障机制。我们自己Run一个Docker镜像，会有许多因素要考虑，例如安全性、网络隔离、日志、性能监控等等。这些K8s都帮我们考虑到了，它提供了一个Docker运行的最佳环境架构，而且还是开源的。

还有，既然我们本地都可以运行K8s的YAML文件，那么我们在云上是不是也能运行？你还记得前面我们留的悬念吧，现在就解决了。

通过K8s，我们要解开云服务商锁定vendor-lock就很简单了。我们只需要将云服务商提供的K8s环境添加到我们kubectl的配置文件中，就可以让我们的应用运行在云服务商的K8s环境中了。目前所有的较大的云服务商都已经加入CNCF，所以当你掌握K8s后，就可以根据云服务商的价格和自己喜好，自由地选择将你的K8s集群部署在CNCF成员的云服务商上，甚至你也可以在自己的机房搭建K8s环境，部署你的应用。

到这儿，我们就部署好了一个K8s集群，那之后我们该如何实时监控容器的运行状态呢？K8s既然能管理容器集群，控制容器运行实例的个数，那应该也能实时监测容器，帮我们解决扩缩容的问题吧？是的，其实上节课我们已经介绍到了容器扩缩容的原理，但并没有给你讲如何实现，那接下来我们就重点看看K8s如何实现实时监控和自动扩缩容。

K8s如何实现扩缩容？

首先，我们要知道的一点就是，K8s其实还向我们隐藏了一部分内容，就是它自身的服务状态。而我们不指定命名空间，默认的命名空间其实是default空间。要查看K8s集群系统的运行状态，我们可以通过指定namespace为kube-system来查看。K8s集群通过namespace隔离，一定程度上，隐藏了系统配置，这可以避免我们误操作。另外它也提供给我们一种逻辑隔离手段，将不同用途的服务和节点划分开来。

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          29h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          29h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          29h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/storage-provisioner                      1/1     Running   0          29h

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   29h

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns   2/2     2            2           29h

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4   2         2         2       29h
```

没错，K8s自己的服务也是运行在自己的集群中的，不过是通过命名空间，将自己做了隔离。这里需要你注意的是，这些服务我不建议你尝试去修改，因为它们涉及到了K8s集群的稳定性；但同时，K8s集群本身也具备扩展性：我们可以通过给K8s安装组件Component，扩展K8s的能力。接下来我先向你介绍K8s中的性能指标metrics组件[5]。

我的代码根目录下已经准备好了metric组件的YAML文件，你只需要执行安装就可以了：

```
kubectl apply -f metrics-components.yaml
```

安装完后，我们再看K8s的系统命名空间：

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          30h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          30h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          30h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/metrics-server-74657b4dc4-t979q          1/1     Running   0          100s
pod/storage-provisioner                      1/1     Running   0          29h

NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns         ClusterIP   10.96.0.10     <none>        53/UDP,53/TCP,9153/TCP   30h
service/metrics-server   ClusterIP   10.97.152.64   <none>        443/TCP                  100s

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns           2/2     2            2           30h
deployment.apps/metrics-server    1/1     1            1           100s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4         2         2         2       30h
replicaset.apps/metrics-server-74657b4dc4  1         1         1       100s
```

对比你就能发现，我们已经安装并启动了metrics-server。那么metrics组件有什么用呢？我们执行下面的命令看看：

```
kubectl top node
```

```
➜  todolist-backend git:(lesson08) kubectl top node
NAME             CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
docker-desktop   316m         7%     1249Mi          66%
```

安装metrics组件后，它就可以将我们应用的监控指标metrics显示出来了。没错，这里我们又可以用到上一讲的内容了。既然我们有了实时的监控指标，那么我们就可以依赖这个指标，来做我们的自动扩缩容了：

```
kubectl autoscale deployment myapp --cpu-percent=30 --min=1 --max=3
```

上面这句话的意思就是，添加一个自动扩缩容部署服务，cpu水位是30%，最小维持1个Pod，最大维持3个Pod。执行完后，我们就发现会多了一个部署服务。

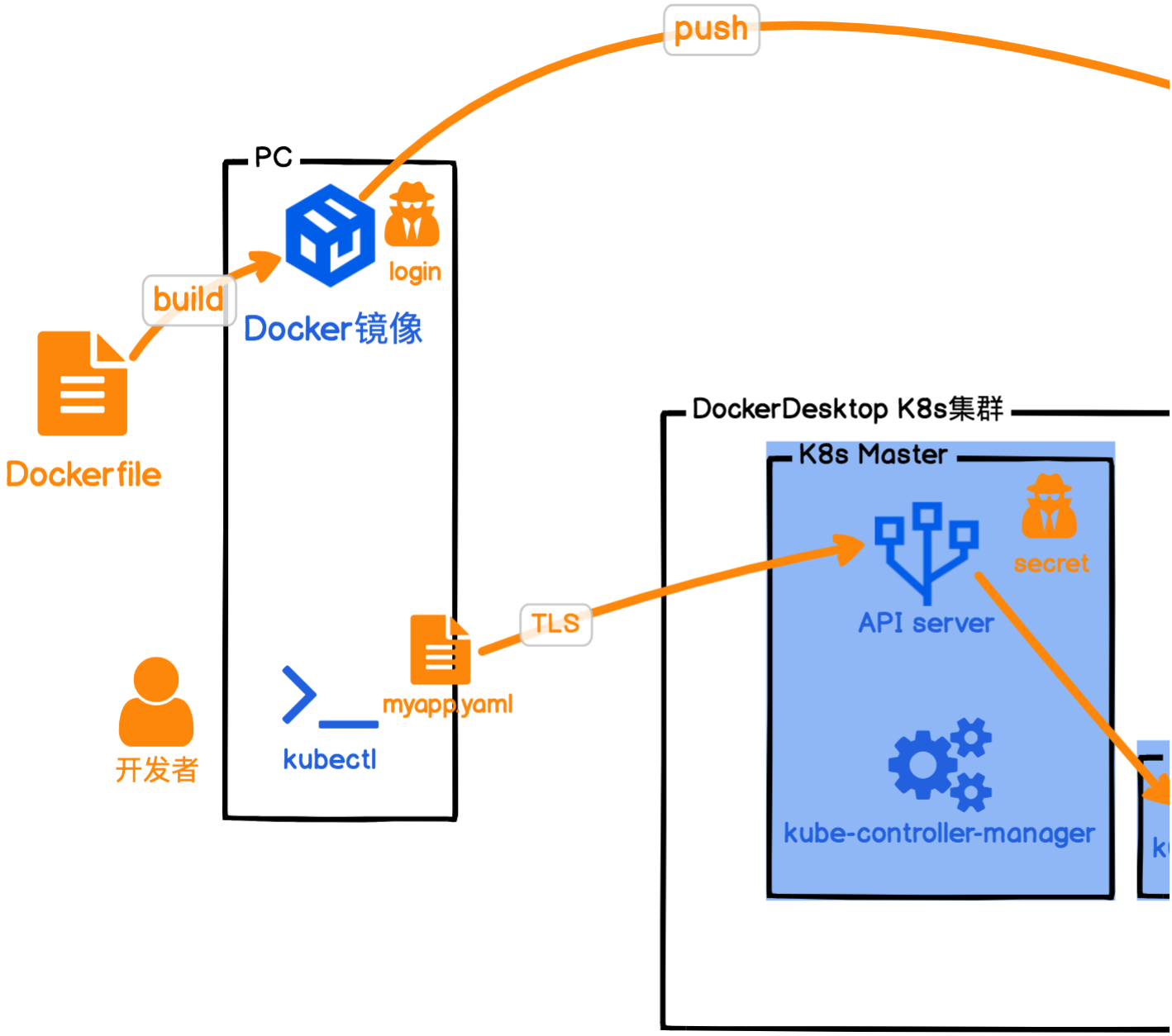
```
kubectl get hpa
```

```
→  todolist-backend git:(lesson08) kubectl get hpa
NAME     REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myapp    Deployment/myapp    0%/30%    1         3         1          22h
```

接下来，我们就可以模拟压测了：

```
kubectl run -i --tty load-generator --image=busybox /bin/sh
$ while true; do wget -q -O- http://10.1.0.16:3001/api/rule; done
```

这里我们用一个K8s的Pod，启动busybox镜像，执行死循环，压测我们的MyApp服务。不过我们目前用Node.js实现的应用可以扛住的流量比较大，单机模拟的压测，轻易还压测不到扩容的水位。



总结

这节课我向你介绍了云原生基金会CNCF的重要成员：Kubernetes。**K8s是用于自动部署、扩展和管理容器化应用程序的开源系统。**云原生其实就是一套通用的云服务商解决方案。

然后我们一起体验了在本地PC上，通过Docker desktop搭建K8s。搭建完后，我还向你介绍了K8s的运行原理：K8s Master节点和Worker节点。其中，Master节点，负责我们整个K8s集群的运作状态；Worker节点则是具体运行我们容器的地方。

之后，我们就开始把“待办任务”Web服务，通过一个K8s的YAML文件来部署，并且暴露NodePort，让我们用浏览器访问。

为了展示K8s如何通过组件Component扩展能力，接着我们介绍了K8s中如何安装使用组件metrics：我们通过一个YAML文件将metrics组件安装到了K8s集群的kube-system命名空间中后，就可以监控到应用的运行指标metrics了。给K8s集群添加上监控指标metrics的能力，我们就可以通过autoscale命令，让应用根据metrics指标和水位来扩缩容了。

最后我们启动了一个BusyBox的Docker容器，模拟压测了我们的“待办任务”Web服务。

总的来说，这节课我们的最终目的就是在本地部署一套K8s集群，通过我们“待办任务”Web服务的K8s版本，让你了解K8s的工作原理。我们已经把下节课的准备工作做好了，下节课我们将在K8s的基础上部署Serverless，可以说，实现属于你自己的Serverless，你已经完成了一半。

作业

这节课是实战课，所以作业就是我们今天要练习的内容。请在你自己的电脑上安装K8s集群，部署我们的“待办任务”Web服务到自己的K8s集群，并从浏览器中访问到K8s集群提供的服务。

另外，你可以尝试一下，手动删除我们部署的MyApp Pod。

```
kubectl delete pod/你的pod名字
```

但你很快就会发现，这个Pod会被K8s重新拉起，而我们要清除部署的MyApp的所有内容其实也很简单，只需要告诉K8s删除我们的myapp.yaml文件创建的资源就可以了。

```
kubectl delete -f myapp.yaml
```

快来动手尝试一下吧，期待你也能分享下今天的成果以及感受。另外，如果今天的内容让你有所收获，也欢迎你把它分享给身边的朋友，邀请他加入学习。

参考资料

[1] https://gitforwindows.org/

[2] https://www.cncf.io/

[3] https://kubernetes.io/zh/

[4] https://github.com/cncf/landscape

[5] https://github.com/kubernetes-incubator/metrics-server/

你好，我是秦粤。上节课我们只是用Docker部署了index.js，如果我们将所有拆解出来的微服务都用Docker独立部署，我们就要同时管理多个Docker容器，也就是Docker集群。如果是更复杂一些的业务，可能需要同时管理几十甚至上百个微服务，显然我们手动维护Docker集群的效率就太低了。而容器即服务CaaS，恰好也需要集群的管理工具。我们也知道FaaS的底层就是CaaS，那CaaS又是如何管理这么多函数实例的呢？怎么做才能提升效率？

我想你应该听过Kubernetes，它也叫K8s（后面统一简称K8s），用于自动部署、扩展和管理容器化应用程序的开源系统，是Docker集群的管理工具。为了解决上述问题，其实我们就可以考虑使用它。K8s的好处就在于，它具备跨环境统一部署的能力。

这节课，我们就试着在本地环境中搭建K8s来管理我们的Docker集群。但正常情况下，这个场景需要几台机器才能完成，而通过Docker，我们还是可以用一台机器就可以在本地搭建一个低配版的K8s。

下节课，我们还会在今天内容的基础上，用K8s的CaaS方式实现一套Serverless环境。通过这两节课的内容，你就可以完整地搭建出属于自己的Serverless了。

话不多说，我们现在就开始，希望你能跟着我一起多动手。

PC上的K8s

那在开始之前，我们先得把安装问题解决了，这部分可能会有点小困难，所以我也给你详细讲下。

首先我们需要安装kubectl，它是K8s的命令行工具。

你需要在你的PC上安装K8s，如果你的操作系统是MacOS或者Windows，那么就比较简单了，桌面版的Docker已经自带了K8s；其它操作系统的同学需要安装minikube。

不过，要顺利启动桌面版Docker自带的K8s，你还得解决国内Docker镜像下载不了的问题，这里请你先下载第8课的代码。接着，请你跟着我的步骤进行操作：

1. 开通阿里云的容器镜像仓库；
2. 在阿里云的容器镜像服务里，找到镜像加速器，复制你的镜像加速器地址；
3. 打开桌面版Docker的控制台，找到Docker Engine。

```
{
  "registry-mirrors" : [
    "https://你的加速地址.mirror.aliyuncs.com"
  ],
  "debug" : true,
  "experimental" : true
}
```

4. 预下载K8s所需要的所有镜像，执行我目录下的docker-k8s-prefetch.sh，如果你是Windows操作系统，建议使用gitBash[1]；

```
chmode +x docker-k8s-prefetch.sh
./docker-k8s-prefetch.sh
```

5. 上面拉取完运行K8s所需的Docker镜像，你就可以在桌面版Docker的K8s项中，勾选启动K8s了。

现在，K8s就能顺利启动了，启动成功后，请你继续执行下面的命令。

查看安装好的K8s系统运行状况。

```
kubectl get all -n kube-system
```

查看K8s有哪些配置环境，对应~/.kube/config。

```
kubectl config get-contexts
```

查看当前K8s环境的整体情况。

```
kubectl get all
```

按照我的流程走，在大部分的机器上本地运行K8s都是没有问题的，如果你卡住了，请在留言区告知我，我帮你解决问题。

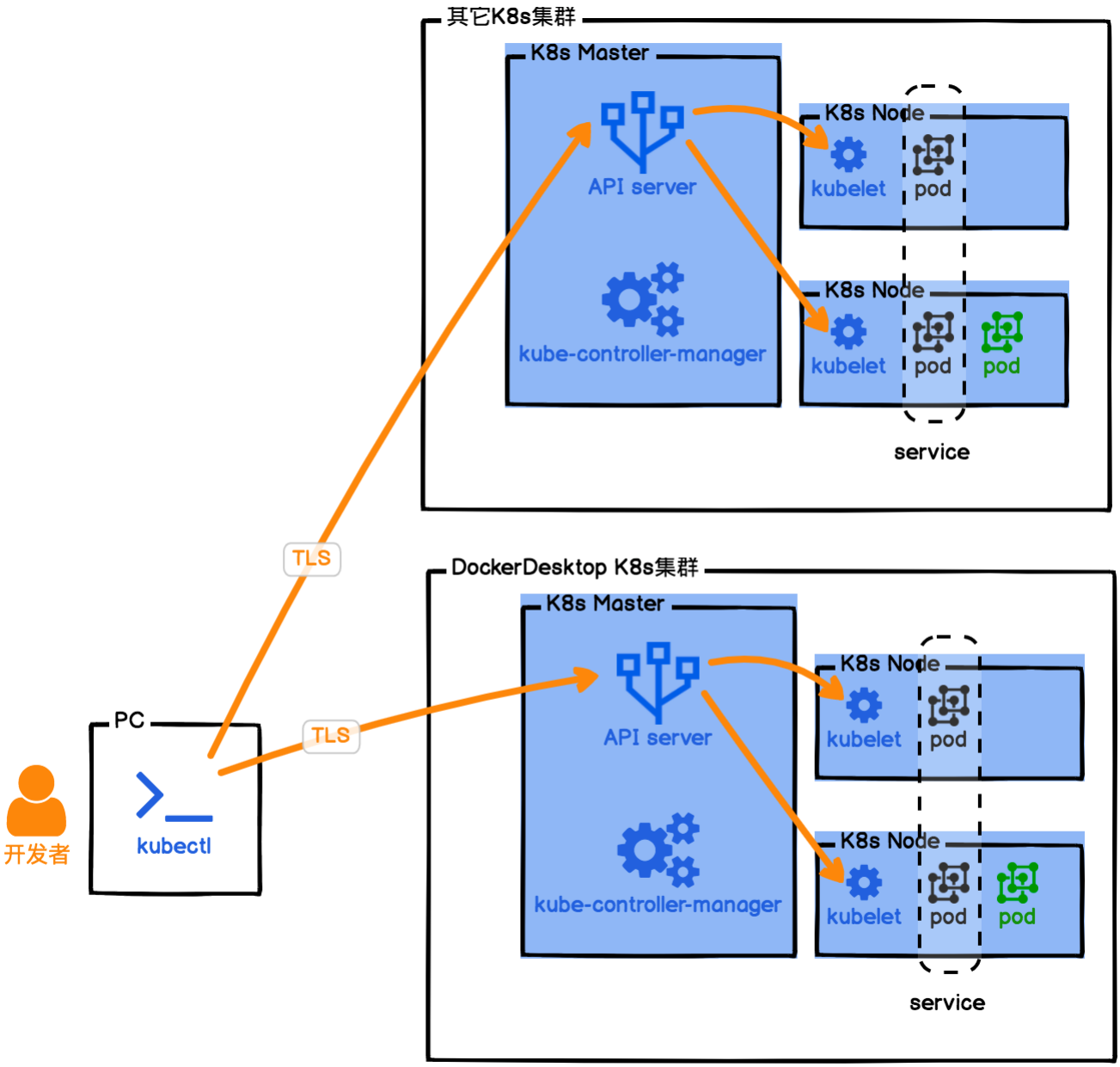
K8s介绍

安装完K8s之后，我们其实还是要简单地了解下K8s，这对于你后面应用它有重要的意义。

我想你应该知道，K8s是云原生Cloud Native[2] 的重要组成部分（目前，K8s的文档[3]已经全面中文化，建议你多翻阅），而云原生其实就是一套通用的云服务商解决方案。在我们的前几节课中，就有同学问：“如果我使用了某个运营商的Serverless，就和这个云服务商强制绑定了怎么办？我是不是就没办法使用其他运营商的服务了？”这就是服务商锁定vendor-lock，而云原生的诞生就是为了解决这个问题，通过云原生基金会CNCF(Cloud Native Computing Foundation)[4]，我们可以得到一整套解锁云服务商的开源解决方案。

那K8s作为云原生Cloud Native的重要组成部分之一，它的作用是什么呢？这里我先留一个悬念，通过理解K8s的原理，你就能清楚这个问题，并充分利用好K8s了。

我们先来看看K8s的原理图：



通过图示我们可以知道，PC本地安装kubectl是K8s的命令行操作工具，通过它，我们就可以控制K8s集群了。又因为kubectl是通过加密通信的，所以我们可以在一台电脑上同时控制多个K8s集群，不过需要指定当前操作的上下文context。这个也就是云原生的重要理念，我们的架构可以部署在多套云服务环境上。

在K8s集群中，Master节点很重要，它是我们整个集群的中枢。没错，Master节点就是Stateful的。Master节点由API Server、etcd、kube-controller-manager等几个核心成员组成，它只负责维持整个K8s集群的状态，为了保证职责单一，Master节点不会运行我们的容器实例。

Worker节点，也就是K8s Node节点，才是我们部署的容器真正运行的地方，但在K8s中，运行容器的最小单位是Pod。一个Pod具备一个集群IP且端口共享，Pod里可以运行一个或多个容器，但最佳的做法还是一个Pod只运行一个容器。这是因为一个Pod里面运行多个容器，容器会竞争Pod的资源，也会影响Pod的启动速度；而一个Pod里只运行一个容器，可以方便我们快速定位问题，监控指标也比较明确。

在K8s集群中，它会构建自己的私有网络，每个容器都有自己的集群IP，容器在集群内部可以互相访问，集群外却无法直接访问。因此我们如果要从外部访问K8s集群提供的服务，则需要通过K8s service将服务暴露出来才行。

案例：“待办任务”K8s版本

现在原理我是讲出来了，但可能还是有点不好理解，接下来我们就还是套进案例去看，依然是我们的“待办任务”Web服务，我们现在把它部署到K8s集群中运行一下，你可以切身体验。相信这样，你就非常清楚这其中的原理了。不过我们本地搭建的例子中，为了节省资源只有一个Master节点，所有的内容都部署在这个Master节点中。

还记得我们上节课构建的Docker镜像吗？我们就用它来部署本地的K8s集群。

我们通常在实际项目中会使用YAML文件来控制我们的应用部署。YAML你可以理解为，就是将我们在K8s部署的所有要做的事情，都写成一个文件，这样就避免了我们要记录大量的kubectl命令执行。不过，K8s也细心地帮我们准备了K8s对象和YAML文件互相转换的能力。这种能力可以让我们快速地将一个K8s集群中部署的结构导出YAML文件，然后再在另一个K8s集群中用这个YAML文件还原出同样的部署结构。

我们需要先确认一下，我们当前的操作是在正确的K8s集群上下文中。对应我们的例子里，也就是看当前选中的集群是不是docker-desktop。

```
kubectl config get-contexts
```

```
→  todolist-backend git:(lesson08) kubectl config get-contexts
CURRENT   NAME                 CLUSTER          AUTHINFO         NAMESPACE
*         docker-desktop       docker-desktop   docker-desktop
          docker-for-desktop   docker-desktop   docker-desktop
          minikube             minikube         minikube
```

如果不对，则需要执行切换集群：

```
kubectl config use-context docker-desktop
```

然后需要我们添加一下拉取镜像的证书服务：

```
kubectl create secret docker-registry regcred --docker-server=registry.cn-shanghai.aliyuncs.com --docker-username=你的容器镜像仓库用户名 --docker-password=你的容器镜像仓库密码
```

这里我需要解释一下，通常我们在镜像仓库中可以设置这个仓库：公开或者私有。如果是操作系统的镜像，设置为公开是完全没有问题的，所有人都可以下载我们的公开镜像；但如果是我们自己的应用镜像，还是需要设置成私有，下载私有镜像需要验证用户身份，也就是Docker Login的操作。因为我们应用的镜像仓库中，包含我们的最终运行代码，往往会有我们数据库的登录用户名和密码，或者我们云服务的ak/sk，这些重要信息如果泄露，很容易让我们的应用受到攻击。

当你添加完secret后，就可以通过下面的命令来查看secret服务了：

```
kubectl get secret regcred
```

另外我还需要再啰嗦一下，secret也不建议你用YAML文件设置，毕竟放着你用户名和密码的文件还是越少越好。

做完准备工作，对我们这次部署的项目而言就很简单了，只需要再执行一句话：

```
kubectl apply -f myapp.yaml
```

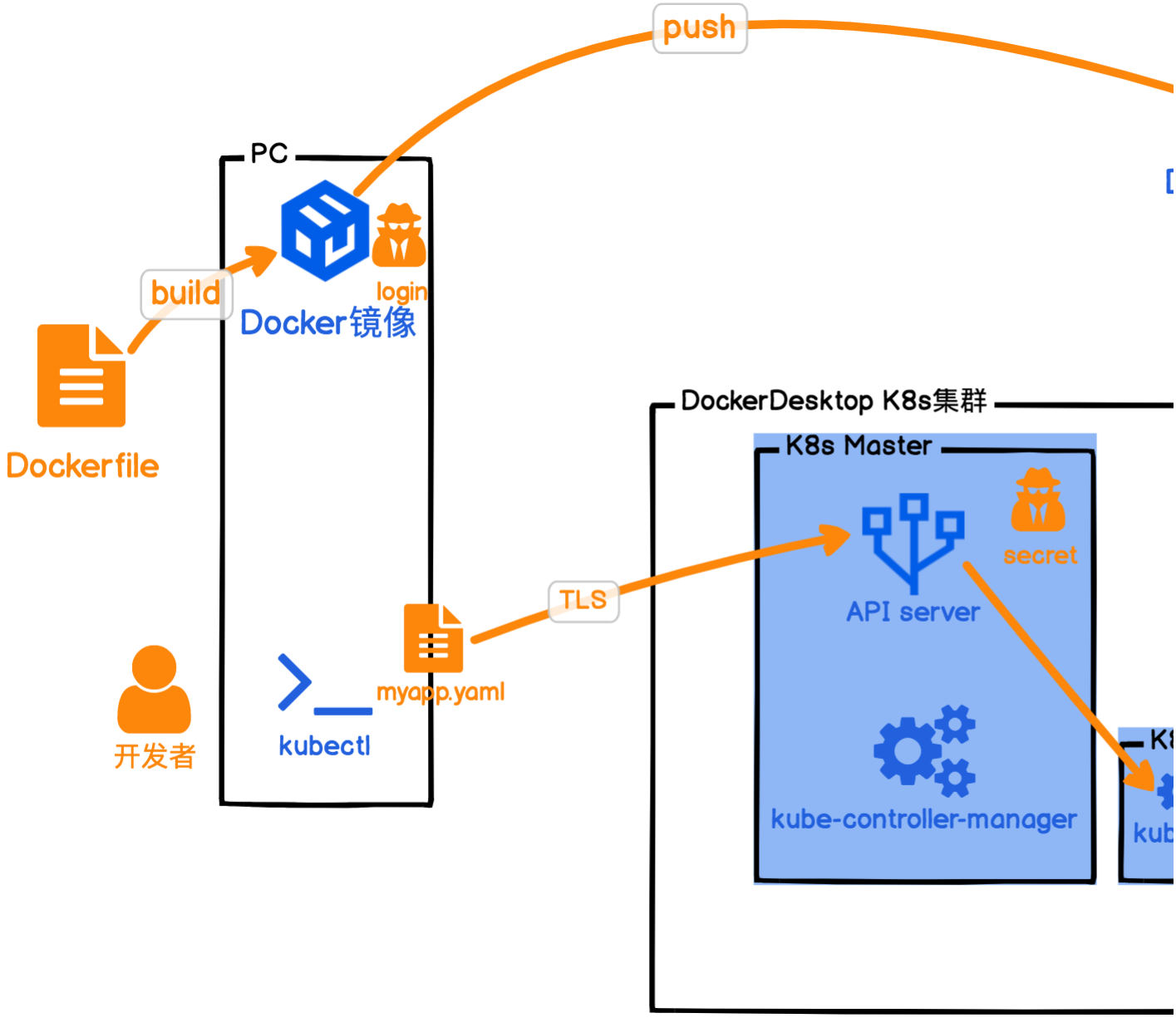
这句话的意思就是，告诉K8s集群，请按照我的部署文件myapp.yaml，部署我的应用。具体如下图所示：

```
➜  todolist-backend git:(lesson08) kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/myapp-759f79bd58-mjwwh   1/1     Running   2          21h

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          29h
service/myapp        NodePort    10.109.56.77   <none>        3001:30512/TCP   21h

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp   1/1     1            1           21h

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-759f79bd58   1         1         1       21h
```

通过获取容器的运行状态，对照上图我粗略地讲解一下我们的myapp.yaml文件吧。

- 首先我们指定要创建一个service/myapp，它选中打了"app:myapp"标签的Pod，集群内访问端口号3001，并且暴露service的端口号30512。
- 然后我们创建了部署服务deployment.apps/myapp，它负责保持我们的Pod数量恒久为1，并且给Pod打上"app:myapp"的标签，也就是负责我们的Pod持久化，一旦Pod挂了，部署服务就会重新拉起一个。
- 最后我们的容器服务，申明了自己的Docker镜像是什么，拉取镜像的secret，以及需要什么资源。

现在我们再回看K8s的原理图，不过这张是实现“待办任务”Web服务版本的：



首先我们可以看出，使用K8s仍然需要我们上节课的Docker发布流程：build、ship、run。不过现在有很多云服务商也会提供Docker镜像构建服务，你只需要上传你的Dockerfile，就会帮你构建镜像并且push到镜像仓库。云服务商提供的镜像构建服务的速度，比你本地构建要快很多倍。

而且相信你也发现了，K8s其实就是一套Docker容器实例的运行保障机制。我们自己Run一个Docker镜像，会有许多因素要考虑，例如安全性、网络隔离、日志、性能监控等等。这些K8s都帮我们考虑到了，它提供了一个Docker运行的最佳环境架构，而且还是开源的。

还有，既然我们本地都可以运行K8s的YAML文件，那么我们在云上是不是也能运行？你还记得前面我们留的悬念吧，现在就解决了。

通过K8s，我们要解开云服务商锁定vendor-lock就很简单了。我们只需要将云服务商提供的K8s环境添加到我们kubectl的配置文件中，就可以让我们的应用运行在云服务商的K8s环境中了。目前所有的较大的云服务商都已经加入CNCF，所以当你掌握K8s后，就可以根据云服务商的价格和自己喜好，自由地选择将你的K8s集群部署在CNCF成员的云服务商上，甚至你也可以在自己的机房搭建K8s环境，部署你的应用。

到这儿，我们就部署好了一个K8s集群，那之后我们该如何实时监控容器的运行状态呢？K8s既然能管理容器集群，控制容器运行实例的个数，那应该也能实时监测容器，帮我们解决扩缩容的问题吧？是的，其实上节课我们已经介绍到了容器扩缩容的原理，但并没有给你讲如何实现，那接下来我们就重点看看K8s如何实现实时监控和自动扩缩容。

K8s如何实现扩缩容？

首先，我们要知道的一点就是，K8s其实还向我们隐藏了一部分内容，就是它自身的服务状态。而我们不指定命名空间，默认的命名空间其实是default空间。要查看K8s集群系统的运行状态，我们可以通过指定namespace为kube-system来查看。K8s集群通过namespace隔离，一定程度上，隐藏了系统配置，这可以避免我们误操作。另外它也提供给我们一种逻辑隔离手段，将不同用途的服务和节点划分开来。

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          29h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          29h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          29h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/storage-provisioner                      1/1     Running   0          29h

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   29h

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns   2/2     2            2           29h

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4   2         2         2       29h
```

没错，K8s自己的服务也是运行在自己的集群中的，不过是通过命名空间，将自己做了隔离。这里需要你注意的是，这些服务我不建议你尝试去修改，因为它们涉及到了K8s集群的稳定性；但同时，K8s集群本身也具备扩展性：我们可以通过给K8s安装组件Component，扩展K8s的能力。接下来我先向你介绍K8s中的性能指标metrics组件[5]。

我的代码根目录下已经准备好了metric组件的YAML文件，你只需要执行安装就可以了：

```
kubectl apply -f metrics-components.yaml
```

安装完后，我们再看K8s的系统命名空间：

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          30h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          30h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          30h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/metrics-server-74657b4dc4-t979q          1/1     Running   0          100s
pod/storage-provisioner                      1/1     Running   0          29h

NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns         ClusterIP   10.96.0.10     <none>        53/UDP,53/TCP,9153/TCP   30h
service/metrics-server   ClusterIP   10.97.152.64   <none>        443/TCP                  100s

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns           2/2     2            2           30h
deployment.apps/metrics-server    1/1     1            1           100s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4         2         2         2       30h
replicaset.apps/metrics-server-74657b4dc4  1         1         1       100s
```

对比你就能发现，我们已经安装并启动了metrics-server。那么metrics组件有什么用呢？我们执行下面的命令看看：

```
kubectl top node
```

```
➜  todolist-backend git:(lesson08) kubectl top node
NAME             CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
docker-desktop   316m         7%     1249Mi          66%
```

安装metrics组件后，它就可以将我们应用的监控指标metrics显示出来了。没错，这里我们又可以用到上一讲的内容了。既然我们有了实时的监控指标，那么我们就可以依赖这个指标，来做我们的自动扩缩容了：

```
kubectl autoscale deployment myapp --cpu-percent=30 --min=1 --max=3
```

上面这句话的意思就是，添加一个自动扩缩容部署服务，cpu水位是30%，最小维持1个Pod，最大维持3个Pod。执行完后，我们就发现会多了一个部署服务。

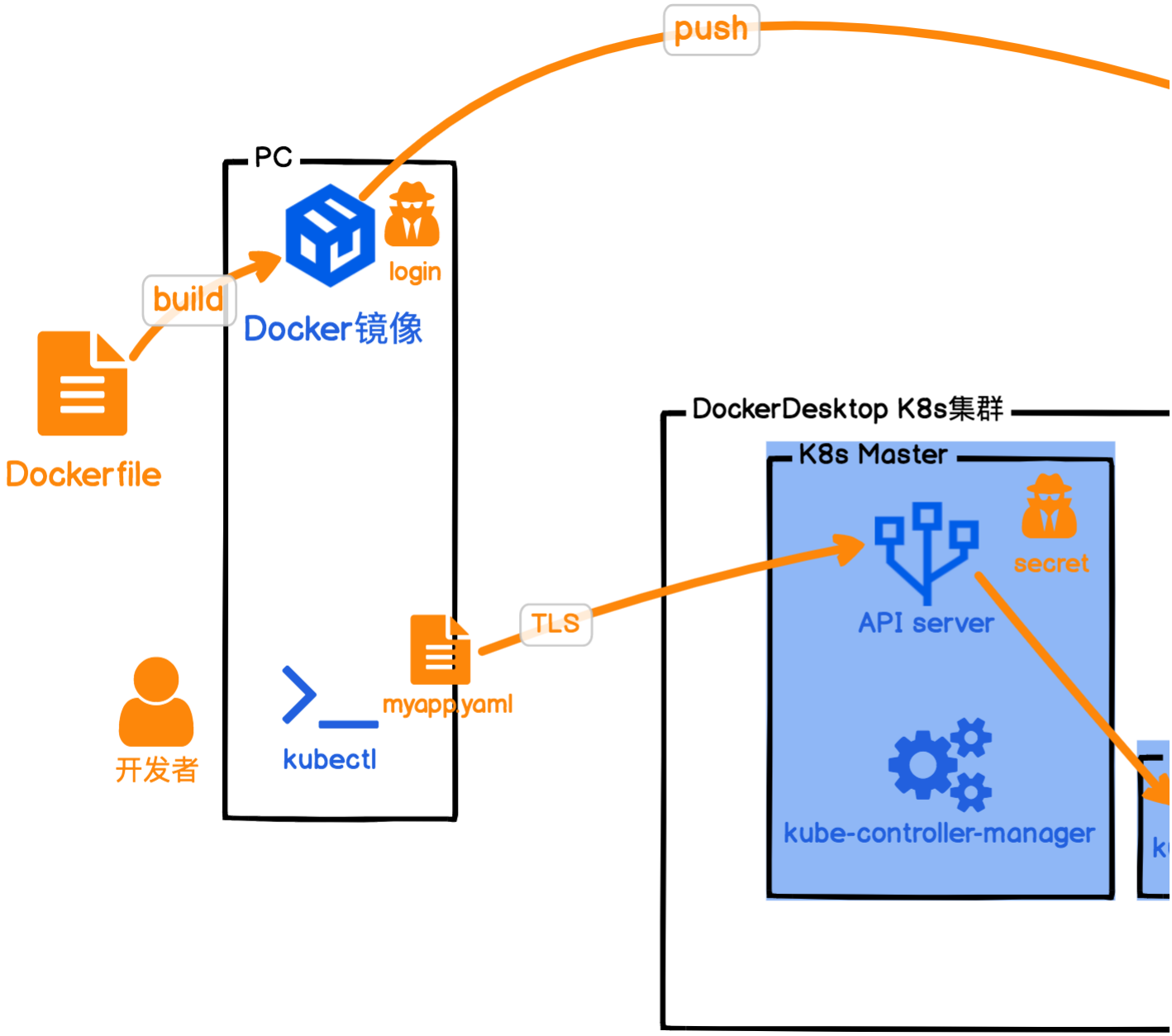
```
kubectl get hpa
```

```
→  todolist-backend git:(lesson08) kubectl get hpa
NAME     REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myapp    Deployment/myapp    0%/30%    1         3         1          22h
```

接下来，我们就可以模拟压测了：

```
kubectl run -i --tty load-generator --image=busybox /bin/sh
$ while true; do wget -q -O- http://10.1.0.16:3001/api/rule; done
```

这里我们用一个K8s的Pod，启动busybox镜像，执行死循环，压测我们的MyApp服务。不过我们目前用Node.js实现的应用可以扛住的流量比较大，单机模拟的压测，轻易还压测不到扩容的水位。



总结

这节课我向你介绍了云原生基金会CNCF的重要成员：Kubernetes。**K8s是用于自动部署、扩展和管理容器化应用程序的开源系统。**云原生其实就是一套通用的云服务商解决方案。

然后我们一起体验了在本地PC上，通过Docker desktop搭建K8s。搭建完后，我还向你介绍了K8s的运行原理：K8s Master节点和Worker节点。其中，Master节点，负责我们整个K8s集群的运作状态；Worker节点则是具体运行我们容器的地方。

之后，我们就开始把“待办任务”Web服务，通过一个K8s的YAML文件来部署，并且暴露NodePort，让我们用浏览器访问。

为了展示K8s如何通过组件Component扩展能力，接着我们介绍了K8s中如何安装使用组件metrics：我们通过一个YAML文件将metrics组件安装到了K8s集群的kube-system命名空间中后，就可以监控到应用的运行指标metrics了。给K8s集群添加上监控指标metrics的能力，我们就可以通过autoscale命令，让应用根据metrics指标和水位来扩缩容了。

最后我们启动了一个BusyBox的Docker容器，模拟压测了我们的“待办任务”Web服务。

总的来说，这节课我们的最终目的就是在本地部署一套K8s集群，通过我们“待办任务”Web服务的K8s版本，让你了解K8s的工作原理。我们已经把下节课的准备工作做好了，下节课我们将在K8s的基础上部署Serverless，可以说，实现属于你自己的Serverless，你已经完成了一半。

作业

这节课是实战课，所以作业就是我们今天要练习的内容。请在你自己的电脑上安装K8s集群，部署我们的“待办任务”Web服务到自己的K8s集群，并从浏览器中访问到K8s集群提供的服务。

另外，你可以尝试一下，手动删除我们部署的MyApp Pod。

```
kubectl delete pod/你的pod名字
```

但你很快就会发现，这个Pod会被K8s重新拉起，而我们要清除部署的MyApp的所有内容其实也很简单，只需要告诉K8s删除我们的myapp.yaml文件创建的资源就可以了。

```
kubectl delete -f myapp.yaml
```

快来动手尝试一下吧，期待你也能分享下今天的成果以及感受。另外，如果今天的内容让你有所收获，也欢迎你把它分享给身边的朋友，邀请他加入学习。

参考资料

[1] https://gitforwindows.org/

[2] https://www.cncf.io/

[3] https://kubernetes.io/zh/

[4] https://github.com/cncf/landscape

[5] https://github.com/kubernetes-incubator/metrics-server/

你好，我是秦粤。上节课我们只是用Docker部署了index.js，如果我们将所有拆解出来的微服务都用Docker独立部署，我们就要同时管理多个Docker容器，也就是Docker集群。如果是更复杂一些的业务，可能需要同时管理几十甚至上百个微服务，显然我们手动维护Docker集群的效率就太低了。而容器即服务CaaS，恰好也需要集群的管理工具。我们也知道FaaS的底层就是CaaS，那CaaS又是如何管理这么多函数实例的呢？怎么做才能提升效率？

我想你应该听过Kubernetes，它也叫K8s（后面统一简称K8s），用于自动部署、扩展和管理容器化应用程序的开源系统，是Docker集群的管理工具。为了解决上述问题，其实我们就可以考虑使用它。K8s的好处就在于，它具备跨环境统一部署的能力。

这节课，我们就试着在本地环境中搭建K8s来管理我们的Docker集群。但正常情况下，这个场景需要几台机器才能完成，而通过Docker，我们还是可以用一台机器就可以在本地搭建一个低配版的K8s。

下节课，我们还会在今天内容的基础上，用K8s的CaaS方式实现一套Serverless环境。通过这两节课的内容，你就可以完整地搭建出属于自己的Serverless了。

话不多说，我们现在就开始，希望你能跟着我一起多动手。

PC上的K8s

那在开始之前，我们先得把安装问题解决了，这部分可能会有点小困难，所以我也给你详细讲下。

首先我们需要安装kubectl，它是K8s的命令行工具。

你需要在你的PC上安装K8s，如果你的操作系统是MacOS或者Windows，那么就比较简单了，桌面版的Docker已经自带了K8s；其它操作系统的同学需要安装minikube。

不过，要顺利启动桌面版Docker自带的K8s，你还得解决国内Docker镜像下载不了的问题，这里请你先下载第8课的代码。接着，请你跟着我的步骤进行操作：

1. 开通阿里云的容器镜像仓库；
2. 在阿里云的容器镜像服务里，找到镜像加速器，复制你的镜像加速器地址；
3. 打开桌面版Docker的控制台，找到Docker Engine。

```
{
  "registry-mirrors" : [
    "https://你的加速地址.mirror.aliyuncs.com"
  ],
  "debug" : true,
  "experimental" : true
}
```

4. 预下载K8s所需要的所有镜像，执行我目录下的docker-k8s-prefetch.sh，如果你是Windows操作系统，建议使用gitBash[1]；

```
chmode +x docker-k8s-prefetch.sh
./docker-k8s-prefetch.sh
```

5. 上面拉取完运行K8s所需的Docker镜像，你就可以在桌面版Docker的K8s项中，勾选启动K8s了。

现在，K8s就能顺利启动了，启动成功后，请你继续执行下面的命令。

查看安装好的K8s系统运行状况。

```
kubectl get all -n kube-system
```

查看K8s有哪些配置环境，对应~/.kube/config。

```
kubectl config get-contexts
```

查看当前K8s环境的整体情况。

```
kubectl get all
```

按照我的流程走，在大部分的机器上本地运行K8s都是没有问题的，如果你卡住了，请在留言区告知我，我帮你解决问题。

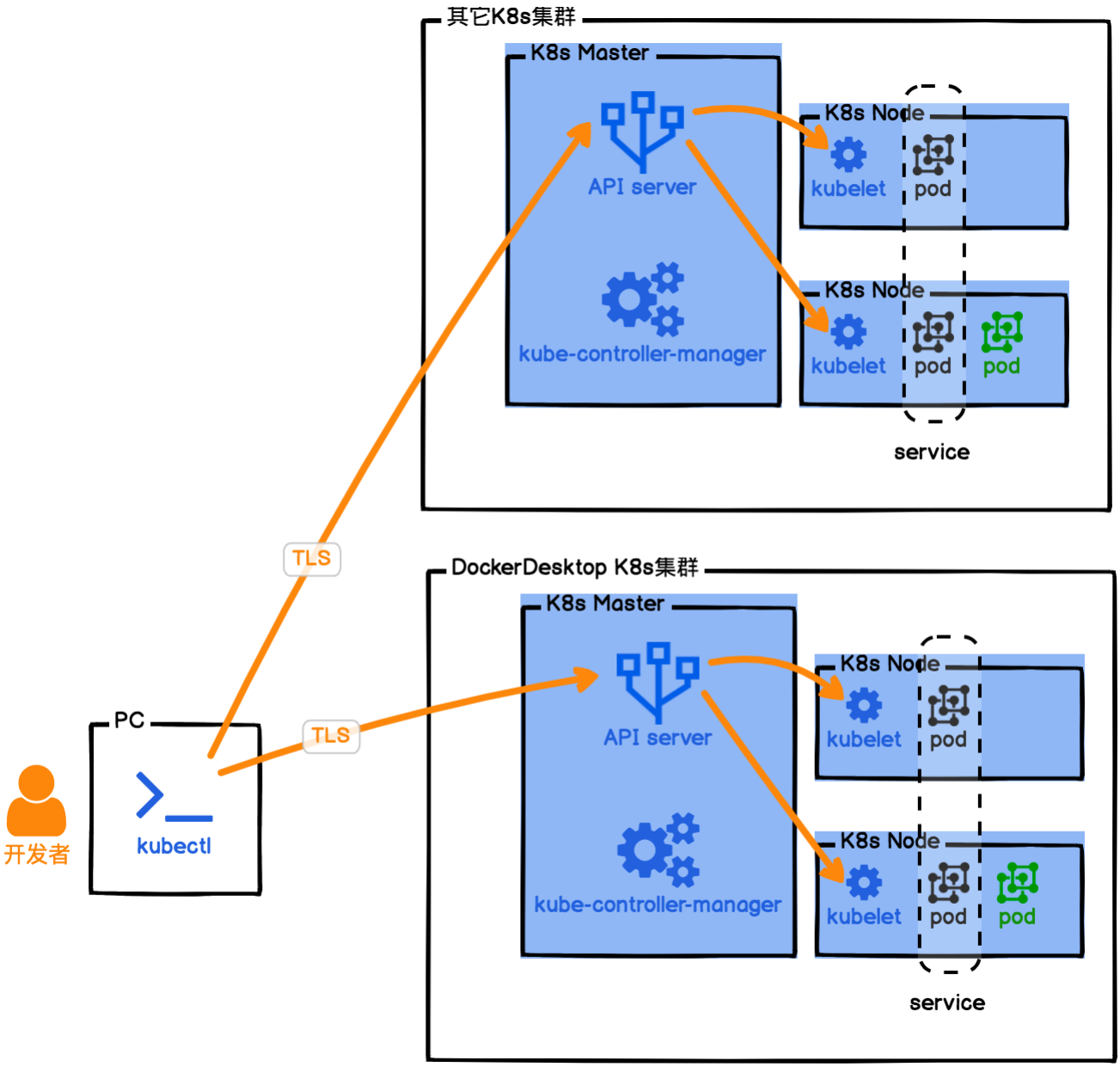
K8s介绍

安装完K8s之后，我们其实还是要简单地了解下K8s，这对于你后面应用它有重要的意义。

我想你应该知道，K8s是云原生Cloud Native[2] 的重要组成部分（目前，K8s的文档[3]已经全面中文化，建议你多翻阅），而云原生其实就是一套通用的云服务商解决方案。在我们的前几节课中，就有同学问：“如果我使用了某个运营商的Serverless，就和这个云服务商强制绑定了怎么办？我是不是就没办法使用其他运营商的服务了？”这就是服务商锁定vendor-lock，而云原生的诞生就是为了解决这个问题，通过云原生基金会CNCF(Cloud Native Computing Foundation)[4]，我们可以得到一整套解锁云服务商的开源解决方案。

那K8s作为云原生Cloud Native的重要组成部分之一，它的作用是什么呢？这里我先留一个悬念，通过理解K8s的原理，你就能清楚这个问题，并充分利用好K8s了。

我们先来看看K8s的原理图：



通过图示我们可以知道，PC本地安装kubectl是K8s的命令行操作工具，通过它，我们就可以控制K8s集群了。又因为kubectl是通过加密通信的，所以我们可以在一台电脑上同时控制多个K8s集群，不过需要指定当前操作的上下文context。这个也就是云原生的重要理念，我们的架构可以部署在多套云服务环境上。

在K8s集群中，Master节点很重要，它是我们整个集群的中枢。没错，Master节点就是Stateful的。Master节点由API Server、etcd、kube-controller-manager等几个核心成员组成，它只负责维持整个K8s集群的状态，为了保证职责单一，Master节点不会运行我们的容器实例。

Worker节点，也就是K8s Node节点，才是我们部署的容器真正运行的地方，但在K8s中，运行容器的最小单位是Pod。一个Pod具备一个集群IP且端口共享，Pod里可以运行一个或多个容器，但最佳的做法还是一个Pod只运行一个容器。这是因为一个Pod里面运行多个容器，容器会竞争Pod的资源，也会影响Pod的启动速度；而一个Pod里只运行一个容器，可以方便我们快速定位问题，监控指标也比较明确。

在K8s集群中，它会构建自己的私有网络，每个容器都有自己的集群IP，容器在集群内部可以互相访问，集群外却无法直接访问。因此我们如果要从外部访问K8s集群提供的服务，则需要通过K8s service将服务暴露出来才行。

案例：“待办任务”K8s版本

现在原理我是讲出来了，但可能还是有点不好理解，接下来我们就还是套进案例去看，依然是我们的“待办任务”Web服务，我们现在把它部署到K8s集群中运行一下，你可以切身体验。相信这样，你就非常清楚这其中的原理了。不过我们本地搭建的例子中，为了节省资源只有一个Master节点，所有的内容都部署在这个Master节点中。

还记得我们上节课构建的Docker镜像吗？我们就用它来部署本地的K8s集群。

我们通常在实际项目中会使用YAML文件来控制我们的应用部署。YAML你可以理解为，就是将我们在K8s部署的所有要做的事情，都写成一个文件，这样就避免了我们要记录大量的kubectl命令执行。不过，K8s也细心地帮我们准备了K8s对象和YAML文件互相转换的能力。这种能力可以让我们快速地将一个K8s集群中部署的结构导出YAML文件，然后再在另一个K8s集群中用这个YAML文件还原出同样的部署结构。

我们需要先确认一下，我们当前的操作是在正确的K8s集群上下文中。对应我们的例子里，也就是看当前选中的集群是不是docker-desktop。

```
kubectl config get-contexts
```

```
→  todolist-backend git:(lesson08) kubectl config get-contexts
CURRENT   NAME                 CLUSTER          AUTHINFO         NAMESPACE
*         docker-desktop       docker-desktop   docker-desktop
          docker-for-desktop   docker-desktop   docker-desktop
          minikube             minikube         minikube
```

如果不对，则需要执行切换集群：

```
kubectl config use-context docker-desktop
```

然后需要我们添加一下拉取镜像的证书服务：

```
kubectl create secret docker-registry regcred --docker-server=registry.cn-shanghai.aliyuncs.com --docker-username=你的容器镜像仓库用户名 --docker-password=你的容器镜像仓库密码
```

这里我需要解释一下，通常我们在镜像仓库中可以设置这个仓库：公开或者私有。如果是操作系统的镜像，设置为公开是完全没有问题的，所有人都可以下载我们的公开镜像；但如果是我们自己的应用镜像，还是需要设置成私有，下载私有镜像需要验证用户身份，也就是Docker Login的操作。因为我们应用的镜像仓库中，包含我们的最终运行代码，往往会有我们数据库的登录用户名和密码，或者我们云服务的ak/sk，这些重要信息如果泄露，很容易让我们的应用受到攻击。

当你添加完secret后，就可以通过下面的命令来查看secret服务了：

```
kubectl get secret regcred
```

另外我还需要再啰嗦一下，secret也不建议你用YAML文件设置，毕竟放着你用户名和密码的文件还是越少越好。

做完准备工作，对我们这次部署的项目而言就很简单了，只需要再执行一句话：

```
kubectl apply -f myapp.yaml
```

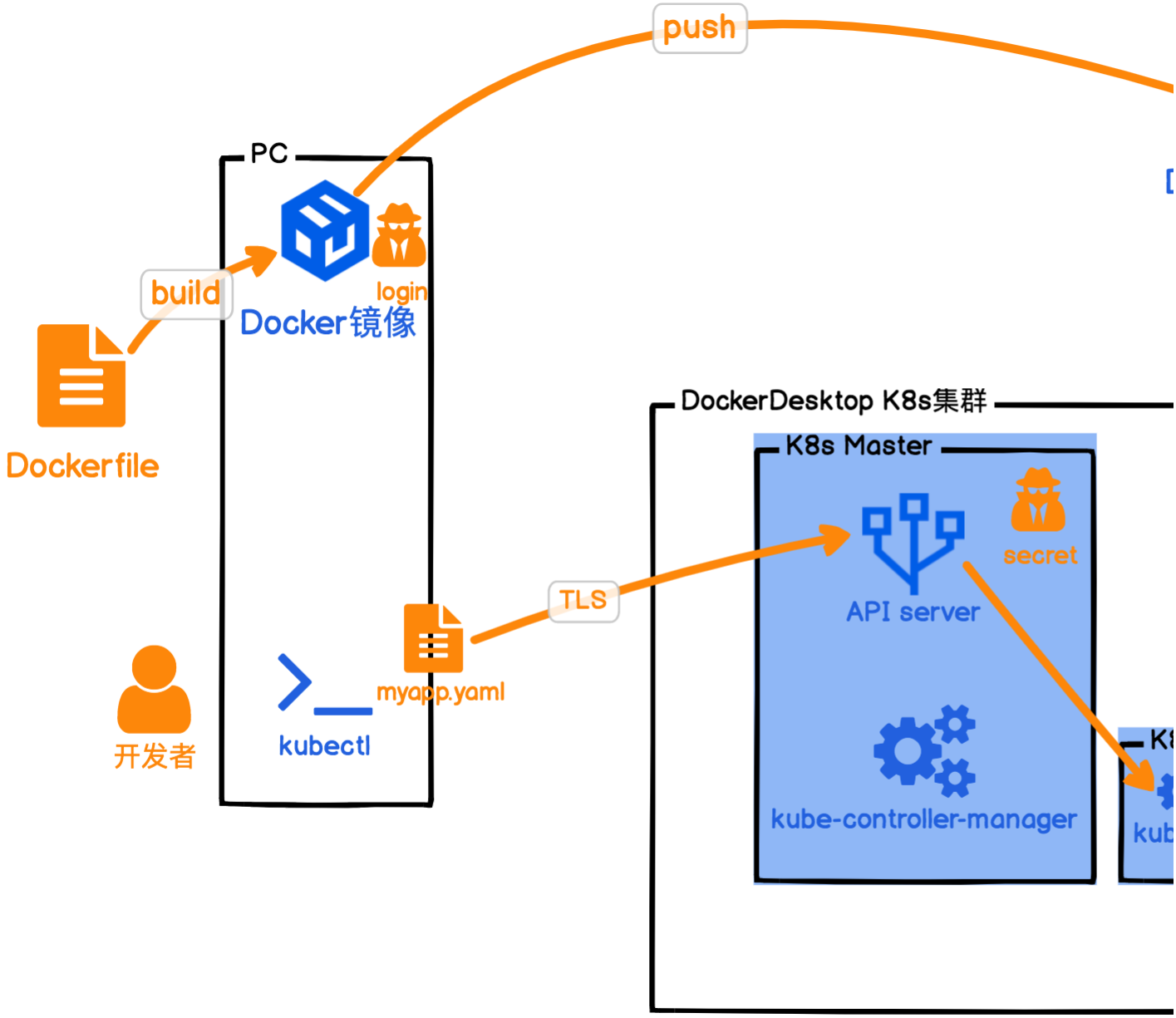
这句话的意思就是，告诉K8s集群，请按照我的部署文件myapp.yaml，部署我的应用。具体如下图所示：

```
➜  todolist-backend git:(lesson08) kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/myapp-759f79bd58-mjwwh   1/1     Running   2          21h

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          29h
service/myapp        NodePort    10.109.56.77   <none>        3001:30512/TCP   21h

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp   1/1     1            1           21h

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-759f79bd58   1         1         1       21h
```

通过获取容器的运行状态，对照上图我粗略地讲解一下我们的myapp.yaml文件吧。

- 首先我们指定要创建一个service/myapp，它选中打了"app:myapp"标签的Pod，集群内访问端口号3001，并且暴露service的端口号30512。
- 然后我们创建了部署服务deployment.apps/myapp，它负责保持我们的Pod数量恒久为1，并且给Pod打上"app:myapp"的标签，也就是负责我们的Pod持久化，一旦Pod挂了，部署服务就会重新拉起一个。
- 最后我们的容器服务，申明了自己的Docker镜像是什么，拉取镜像的secret，以及需要什么资源。

现在我们再回看K8s的原理图，不过这张是实现“待办任务”Web服务版本的：



首先我们可以看出，使用K8s仍然需要我们上节课的Docker发布流程：build、ship、run。不过现在有很多云服务商也会提供Docker镜像构建服务，你只需要上传你的Dockerfile，就会帮你构建镜像并且push到镜像仓库。云服务商提供的镜像构建服务的速度，比你本地构建要快很多倍。

而且相信你也发现了，K8s其实就是一套Docker容器实例的运行保障机制。我们自己Run一个Docker镜像，会有许多因素要考虑，例如安全性、网络隔离、日志、性能监控等等。这些K8s都帮我们考虑到了，它提供了一个Docker运行的最佳环境架构，而且还是开源的。

还有，既然我们本地都可以运行K8s的YAML文件，那么我们在云上是不是也能运行？你还记得前面我们留的悬念吧，现在就解决了。

通过K8s，我们要解开云服务商锁定vendor-lock就很简单了。我们只需要将云服务商提供的K8s环境添加到我们kubectl的配置文件中，就可以让我们的应用运行在云服务商的K8s环境中了。目前所有的较大的云服务商都已经加入CNCF，所以当你掌握K8s后，就可以根据云服务商的价格和自己喜好，自由地选择将你的K8s集群部署在CNCF成员的云服务商上，甚至你也可以在自己的机房搭建K8s环境，部署你的应用。

到这儿，我们就部署好了一个K8s集群，那之后我们该如何实时监控容器的运行状态呢？K8s既然能管理容器集群，控制容器运行实例的个数，那应该也能实时监测容器，帮我们解决扩缩容的问题吧？是的，其实上节课我们已经介绍到了容器扩缩容的原理，但并没有给你讲如何实现，那接下来我们就重点看看K8s如何实现实时监控和自动扩缩容。

K8s如何实现扩缩容？

首先，我们要知道的一点就是，K8s其实还向我们隐藏了一部分内容，就是它自身的服务状态。而我们不指定命名空间，默认的命名空间其实是default空间。要查看K8s集群系统的运行状态，我们可以通过指定namespace为kube-system来查看。K8s集群通过namespace隔离，一定程度上，隐藏了系统配置，这可以避免我们误操作。另外它也提供给我们一种逻辑隔离手段，将不同用途的服务和节点划分开来。

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          29h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          29h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          29h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/storage-provisioner                      1/1     Running   0          29h

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   29h

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns   2/2     2            2           29h

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4   2         2         2       29h
```

没错，K8s自己的服务也是运行在自己的集群中的，不过是通过命名空间，将自己做了隔离。这里需要你注意的是，这些服务我不建议你尝试去修改，因为它们涉及到了K8s集群的稳定性；但同时，K8s集群本身也具备扩展性：我们可以通过给K8s安装组件Component，扩展K8s的能力。接下来我先向你介绍K8s中的性能指标metrics组件[5]。

我的代码根目录下已经准备好了metric组件的YAML文件，你只需要执行安装就可以了：

```
kubectl apply -f metrics-components.yaml
```

安装完后，我们再看K8s的系统命名空间：

```
➜  todolist-backend git:(lesson08) kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/coredns-5c98db65d4-4rn67                 1/1     Running   1          30h
pod/coredns-5c98db65d4-xd2fz                 1/1     Running   1          30h
pod/etcd-docker-desktop                      1/1     Running   0          29h
pod/kube-apiserver-docker-desktop            1/1     Running   0          29h
pod/kube-controller-manager-docker-desktop   1/1     Running   0          29h
pod/kube-proxy-d9kts                         1/1     Running   0          30h
pod/kube-scheduler-docker-desktop            1/1     Running   0          29h
pod/metrics-server-74657b4dc4-t979q          1/1     Running   0          100s
pod/storage-provisioner                      1/1     Running   0          29h

NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns         ClusterIP   10.96.0.10     <none>        53/UDP,53/TCP,9153/TCP   30h
service/metrics-server   ClusterIP   10.97.152.64   <none>        443/TCP                  100s

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
daemonset.apps/kube-proxy   1         1         1       1            1           beta.kubernetes.io/o

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns           2/2     2            2           30h
deployment.apps/metrics-server    1/1     1            1           100s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-5c98db65d4         2         2         2       30h
replicaset.apps/metrics-server-74657b4dc4  1         1         1       100s
```

对比你就能发现，我们已经安装并启动了metrics-server。那么metrics组件有什么用呢？我们执行下面的命令看看：

```
kubectl top node
```

```
➜  todolist-backend git:(lesson08) kubectl top node
NAME             CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
docker-desktop   316m         7%     1249Mi          66%
```

安装metrics组件后，它就可以将我们应用的监控指标metrics显示出来了。没错，这里我们又可以用到上一讲的内容了。既然我们有了实时的监控指标，那么我们就可以依赖这个指标，来做我们的自动扩缩容了：

```
kubectl autoscale deployment myapp --cpu-percent=30 --min=1 --max=3
```

上面这句话的意思就是，添加一个自动扩缩容部署服务，cpu水位是30%，最小维持1个Pod，最大维持3个Pod。执行完后，我们就发现会多了一个部署服务。

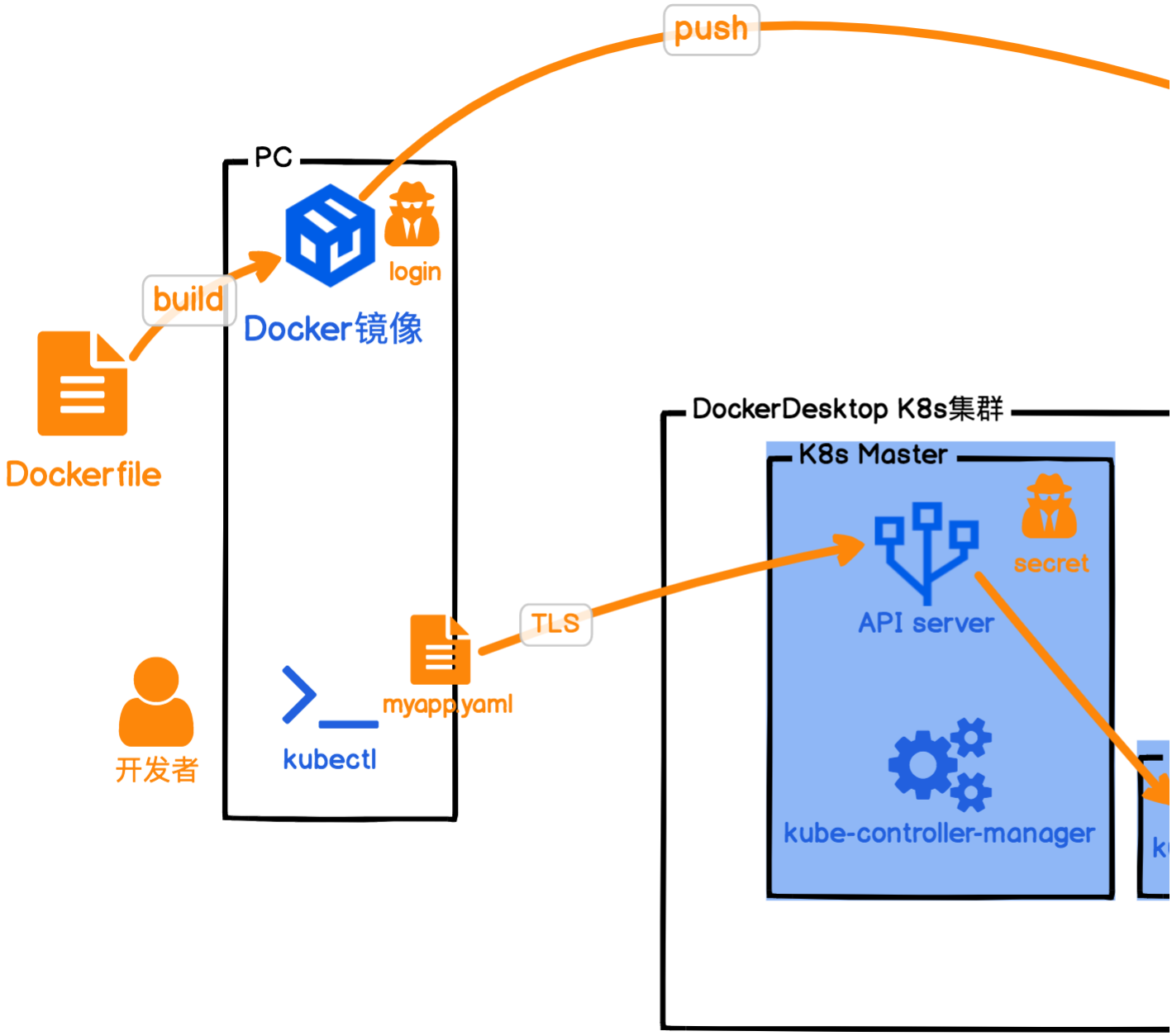
```
kubectl get hpa
```

```
→  todolist-backend git:(lesson08) kubectl get hpa
NAME     REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myapp    Deployment/myapp    0%/30%    1         3         1          22h
```

接下来，我们就可以模拟压测了：

```
kubectl run -i --tty load-generator --image=busybox /bin/sh
$ while true; do wget -q -O- http://10.1.0.16:3001/api/rule; done
```

这里我们用一个K8s的Pod，启动busybox镜像，执行死循环，压测我们的MyApp服务。不过我们目前用Node.js实现的应用可以扛住的流量比较大，单机模拟的压测，轻易还压测不到扩容的水位。



总结

这节课我向你介绍了云原生基金会CNCF的重要成员：Kubernetes。**K8s是用于自动部署、扩展和管理容器化应用程序的开源系统。**云原生其实就是一套通用的云服务商解决方案。

然后我们一起体验了在本地PC上，通过Docker desktop搭建K8s。搭建完后，我还向你介绍了K8s的运行原理：K8s Master节点和Worker节点。其中，Master节点，负责我们整个K8s集群的运作状态；Worker节点则是具体运行我们容器的地方。

之后，我们就开始把“待办任务”Web服务，通过一个K8s的YAML文件来部署，并且暴露NodePort，让我们用浏览器访问。

为了展示K8s如何通过组件Component扩展能力，接着我们介绍了K8s中如何安装使用组件metrics：我们通过一个YAML文件将metrics组件安装到了K8s集群的kube-system命名空间中后，就可以监控到应用的运行指标metrics了。给K8s集群添加上监控指标metrics的能力，我们就可以通过autoscale命令，让应用根据metrics指标和水位来扩缩容了。

最后我们启动了一个BusyBox的Docker容器，模拟压测了我们的“待办任务”Web服务。

总的来说，这节课我们的最终目的就是在本地部署一套K8s集群，通过我们“待办任务”Web服务的K8s版本，让你了解K8s的工作原理。我们已经把下节课的准备工作做好了，下节课我们将在K8s的基础上部署Serverless，可以说，实现属于你自己的Serverless，你已经完成了一半。

作业

这节课是实战课，所以作业就是我们今天要练习的内容。请在你自己的电脑上安装K8s集群，部署我们的“待办任务”Web服务到自己的K8s集群，并从浏览器中访问到K8s集群提供的服务。

另外，你可以尝试一下，手动删除我们部署的MyApp Pod。

```
kubectl delete pod/你的pod名字
```

但你很快就会发现，这个Pod会被K8s重新拉起，而我们要清除部署的MyApp的所有内容其实也很简单，只需要告诉K8s删除我们的myapp.yaml文件创建的资源就可以了。

```
kubectl delete -f myapp.yaml
```

快来动手尝试一下吧，期待你也能分享下今天的成果以及感受。另外，如果今天的内容让你有所收获，也欢迎你把它分享给身边的朋友，邀请他加入学习。

参考资料

[1] https://gitforwindows.org/

[2] https://www.cncf.io/

[3] https://kubernetes.io/zh/

[4] https://github.com/cncf/landscape

[5] https://github.com/kubernetes-incubator/metrics-server/