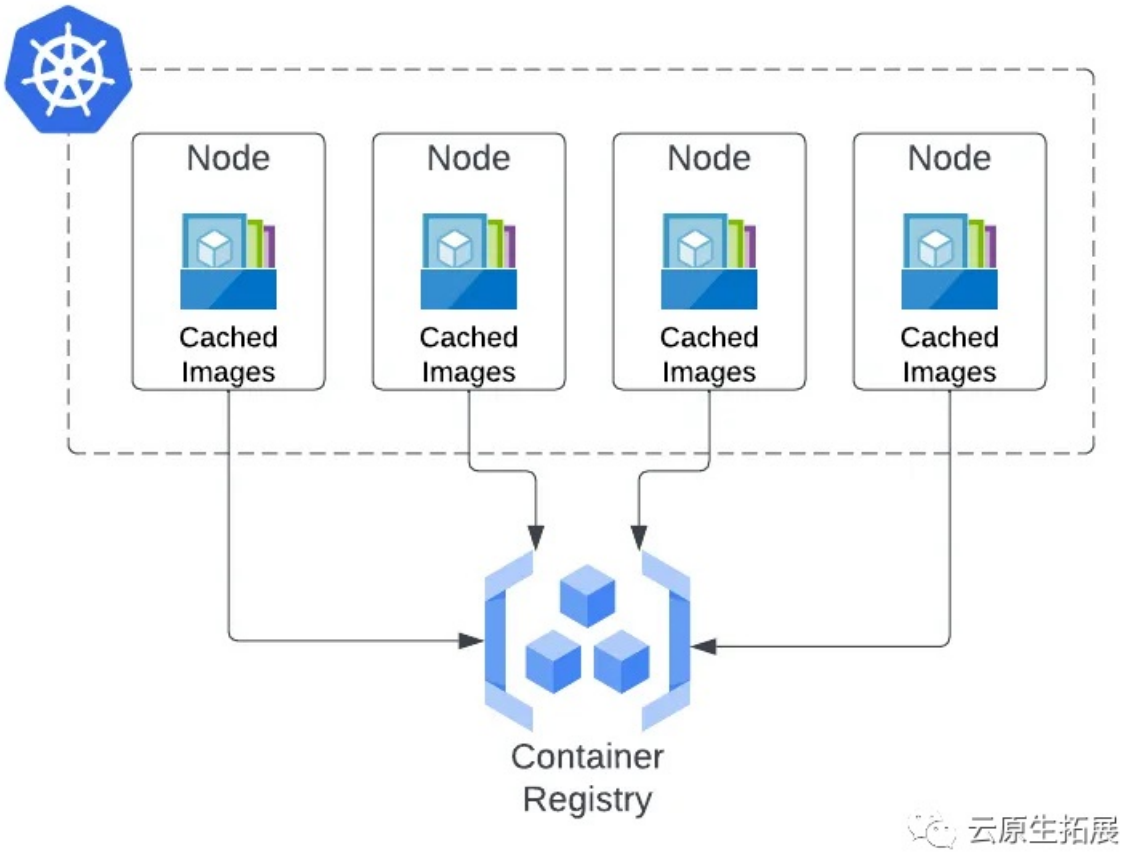
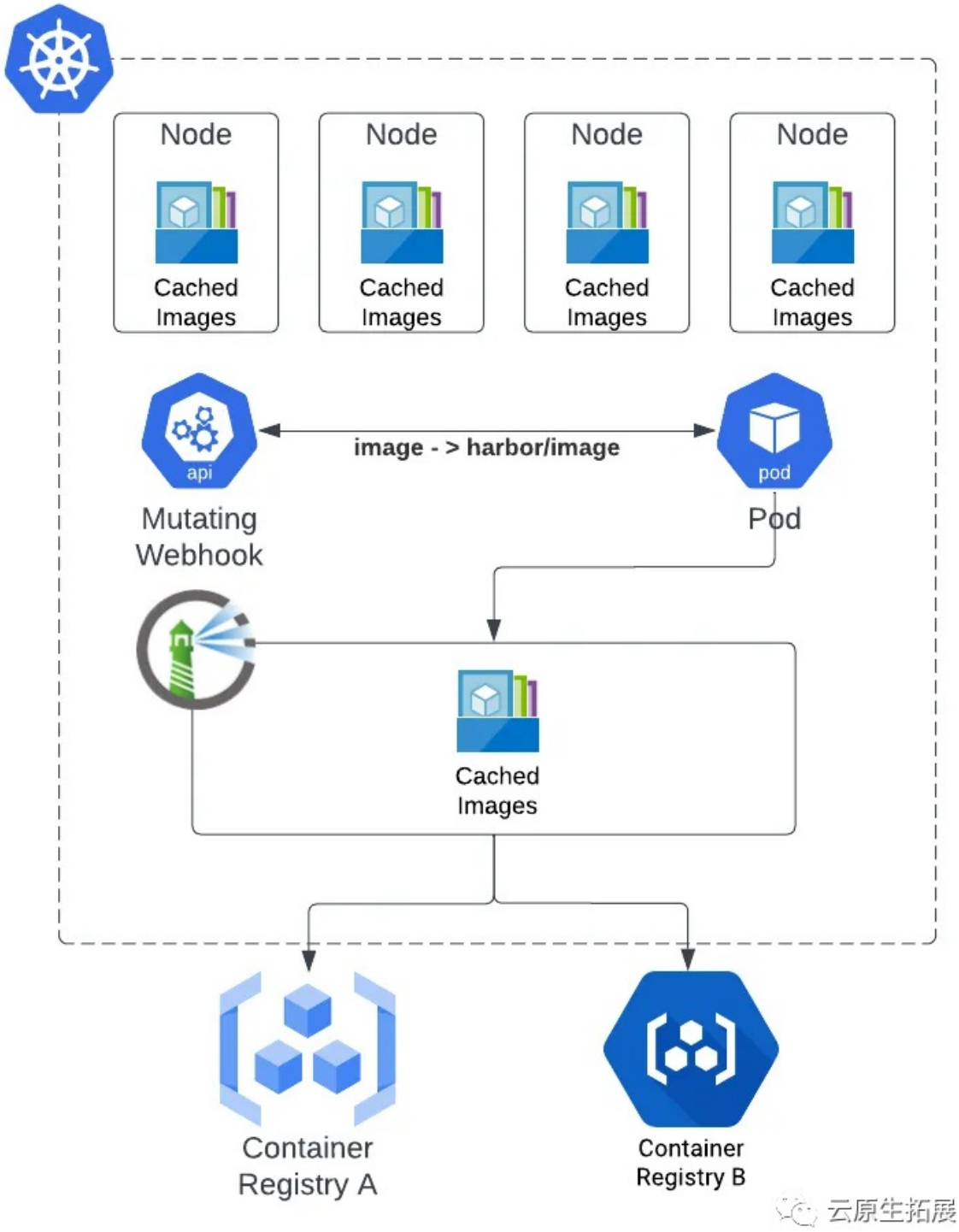
118Kubernetes 系列（一一一）Kubernetes Harbor 镜像代理缓存 — 从几分钟到几毫秒

当你将工作负载部署到 Kubernetes 时，某个 Pod 中的容器自然是基于 OCI 容器镜像的。这些镜像可以从多种私人/公共存储库中提取。Kubernetes 在每个拉取镜像的节点上本地缓存镜像，以便其他 Pod 可以使用相同的镜像。Kubernetes 如何以及何时拉取镜像的设置可以在文档（https://kubernetes.io/docs/concepts/containers/images/）中找到。



在大多数用例中，这还不够。如今，大多数 Cloud Kubernetes 集群都需要自动扩展，并根据客户的使用情况动态分配节点。如果多个节点必须多次拉取同一个镜像怎么办？如果此镜像很重，则可能需要几分钟的时间。在应用程序自动缩放领域，这是一个相对较长的时间。

解决方案



预期的解决方案需要在 Kubernetes 之上有一个缓存层，这样 Kubernetes 就有一个集中的镜像缓存，所有节点都从它“拉取”。但由于缓存需要非常快，因此缓存解决方案需要位于 Kubernetes 内部，并且所有节点都应具有最快的延迟。

Harbor 是一个 CNCF 毕业项目，充当容器参控股，但最重要的是充当拉取代理缓存。

拉取代理缓存是一种缓存机制，旨在优化容器仓库环境中容器镜像的分发和检索。它充当客户端（例如容器运行时或构建系统）和上游容器仓库之间的中介。

当客户端请求容器镜像时，代理缓存会检查是否已有所请求镜像的本地副本。如果镜像存在，代理缓存会直接将其提供给客户端，从而无需从上游仓库下载它。这可以减少网络延迟并节省带宽。

如果本地缓存中不存在所请求的镜像，则代理缓存将充当常规代理并将请求转发到上游仓库。然后，代理缓存从参控股中检索镜像并将其提供给客户端。此外，代理缓存还会在其本地缓存中存储镜像的副本以供将来请求。

这种缓存机制具有多种优点，包括提高性能、减少网络流量和增强可靠性。通过减少对上游仓库的依赖，可以显著加快容器化环境中容器镜像的分发和部署速度。

在我们的例子中，Harbor Pull Through Proxy Cache 位于一个（或多个）本地 Kubernetes 节点上，这些节点共享网络并且在延迟方面接近所有其他节点。这基本上意味着，每个节点不是从互联网/远程拉取，而是从另一个节点拉取。这本质上意味着来自远程的拉取发生在“一个节点——一次”，而不是“多个节点——多次”。

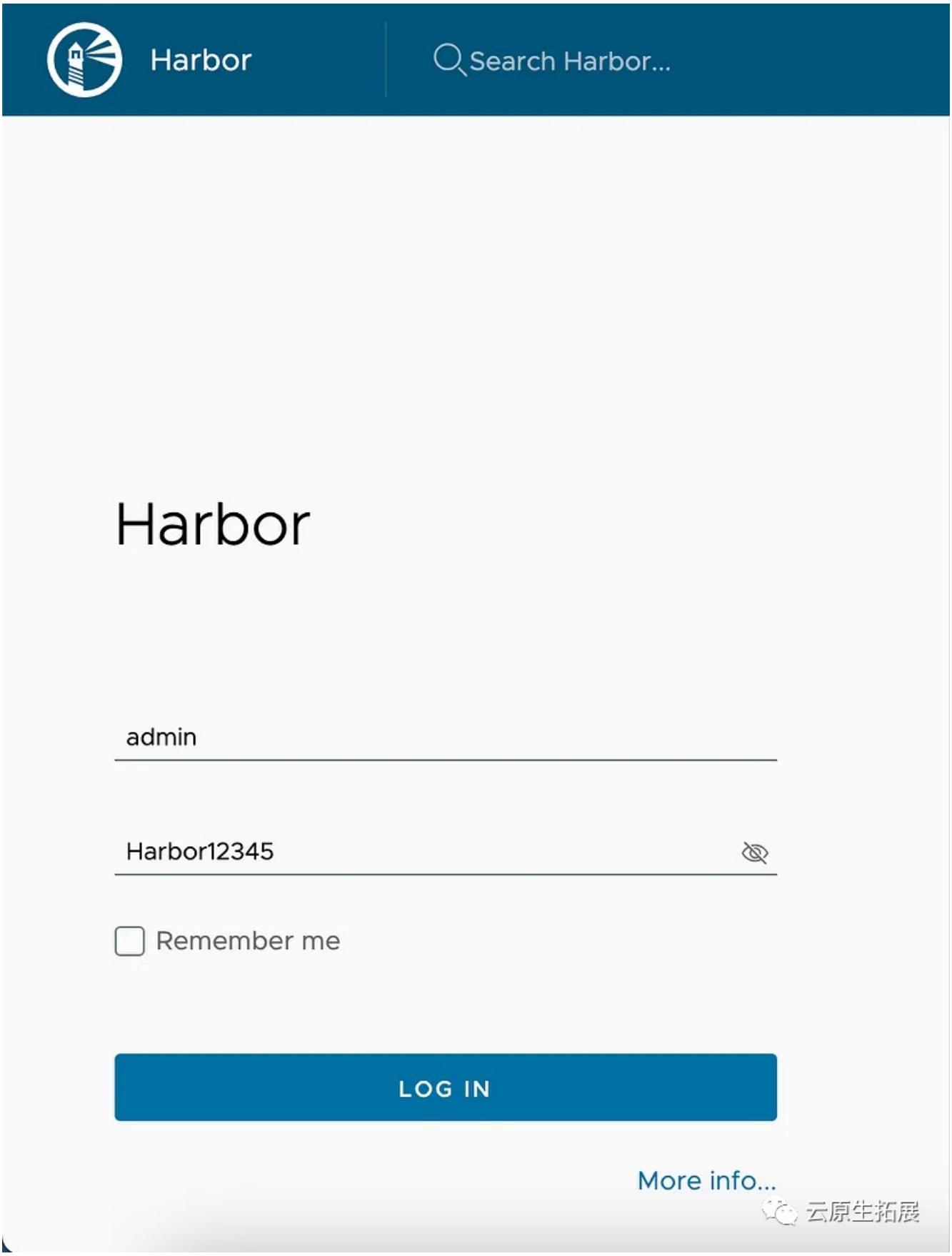
方法

需要在 Kubernetes 上设置的第一个组件是 Harbor。为此，我们可以使用官方的 Bitnami Harbor Helm Chart（https://artifacthub.io/packages/helm/bitnami/harbor）。Values 必须包括 Ingress、镜像缓存的持久化（如果您想使用它）以及我发现最适合此用例的一些其他设置（您应该包括 commonLabels 中提供的标签，稍后将对此进行解释）。

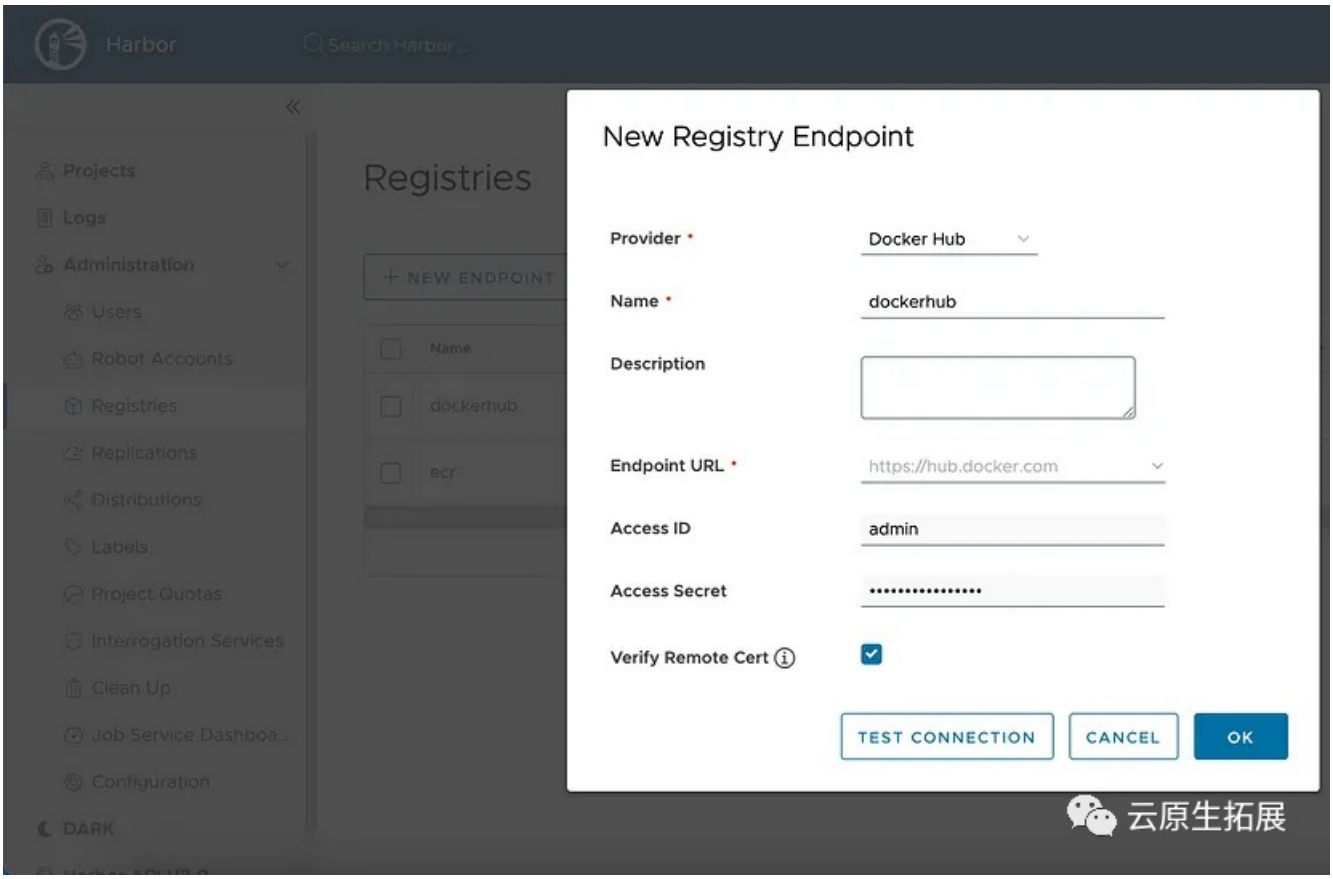
```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install my-harbor bitnami/harbor
```

```
externalURL: https://my-harbor.my.domain
exposureType: ingress
adminPassword: Harbor12345
commonLabels:
  goharbor.io/harbor-container-webhook-disable: 'true'
ingress:
  core:
    hostname: my-harbor.my.domain
persistence:
  enabled: true
  persistentVolumeClaim:
    registry:
      size: 800Gi # == 100GB
      accessModes:
        - ReadWriteOnce
postgresql:
  architecture: replication
  readReplicas:
    extendedConfiguration: |
      max_connections = 1024
  primary:
    extendedConfiguration: |
      max_connections = 1024
```

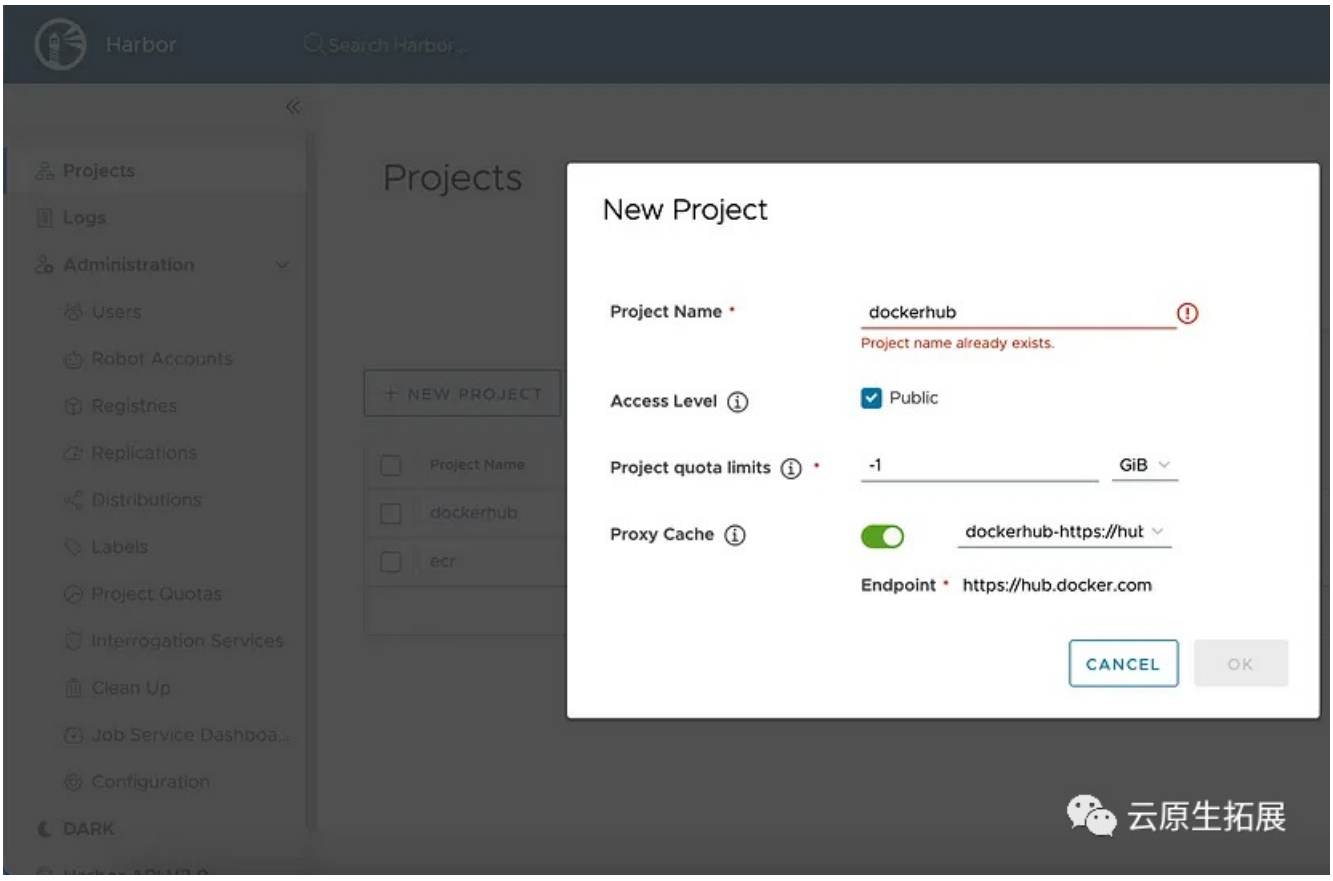
安装完成后，您应该使用默认凭据登录到您提供的 ingress 端点：



接下来，您应该配置您的第一个仓库端点。在这种情况下，您可以放置您使用的任何仓库，但我们将使用我们拥有的帐户的用户名/密码放置 DockerHub 提供程序：



接下来，您应该创建一个指向仓库（代理缓存）的项目。就我而言，它已经存在。您应该根据您的用例定义公共或不公共。这定义了该项目本身在拉/推镜像时是否需要权限。



现在，从技术上讲，您应该能够通过您的 Harbor dockerhub 项目拉取任何 dockerhub 镜像。它看起来像这样。

代替：

```
docker pull redis:latest
```

你可以执行以下操作：

```
docker pull my-harbor.my.domain/dockerhub/redis:latest
```

您现在可以享受代理缓存的好处。下一阶段回答的问题是：“我们如何让 Kubernetes 自动使用这个代理缓存”？

Harbor 缓存 Mutating Webhook

在此存储库(https://github.com/indeedeng-alpha/harbor-container-webhook)中，您将找到一个harbor-container-webhook项目。如果您不熟悉，Kubernetes mutating Webhooks 是 API 扩展，您可以提供 Kubernetes 调度机制，以便在应用和配置对象之前实时更改对象。Mutating Webhook 包含 Kubernetes 使用的 API 服务中公开的自定义逻辑。

在我们的例子中，这个 webhook 将“转换”具有特定前缀/逻辑的所有镜像以使用harbor而不是其原始存储库。这是不假思索的，这意味着您仍然会应用您的 pod，假设它会到达远程存储库，但在幕后它将使用 Harbor 代理缓存。

通过 Helm 安装的此 Webhook 的值可以包含规则。这些规则基于正则表达式，让您可以控制要变异的镜像。（您应该注意到我排除了 Harbor 本身，因为如果 Harbor 关闭，Harbor 就无法使用 Harbor）。

```
rules:
  - name: 'docker.io rewrite rule'
    matches:
      - '^docker.io'
    excludes:
      - '.*goharbor.*'
    replace: 'my-harbor.my-domain/dockerhub'
    checkUpstream: true
```

您还可以使用命名空间和 Pod 中的标签添加保护，这将允许您精细地添加此功能，而不是一次全部添加。例如：

```
webhook:
  namespaceSelector:
    matchExpressions:
      - key: "goharbor.io/harbor-container-webhook-enable"
        operator: In
        values: ["true"]
  objectSelector:
    matchExpressions:
      - key: "goharbor.io/harbor-container-webhook-disable"
        operator: NotIn
        values: ["true"]
```

现在，您拥有一个完全自动化的 Kubernetes 镜像缓存，该缓存尽可能靠近您的节点。作为证明，我的工作负载镜像拉取时间大规模地从几秒减少到几毫秒！

过去：

```
Successfully pulled image 182893536443 dkr.ecr.us east 1 amazonaws.com em:em2_prod_eds group_deploy em enrichment_3298 in 46.787790789s
Successfully pulled image 182893536443 dkr.ecr.us east 1 amazonaws.com em:em2_prod_eds group_deploy em enrichment_3298 in 46.814690842s
Successfully pulled image 182893536443 dkr.ecr.us east 1 amazonaws.com em:em2_prod_eds group_deploy em enrich
Successfully pulled image 182893536443 dkr.ecr.us east 1 amazonaws.com em:em2_prod_eds group_deploy em enrichment_3298 in 47.27677886s 4
```

现在：

```
Successfully pulled image harbor platform prod us.int.explorium.ninja ecr em:em2_prod_eds group_deploy em enrichment_3298 in 146.735912ms
Successfully pulled image harbor platform prod us.int.explorium.ninja ecr em:em2_prod_eds group_deploy em enrichment_3298 in 146.989553ms
Successfully pulled image harbor platform prod us.int.explorium.ninja ecr em:em2_prod_eds group_deploy em enrich
Successfully pulled image harbor platform prod us.int.explorium.ninja ecr em:em2_prod_eds group_deploy em enrichment_3298 in 151.500729ms
```