和鲸

Kubernetes预测性集群伸缩

HeyWhale.com
上 海 和 今 信 息 科 技 有 限 公 司

CONTENTS
目录{

{ 1 }

Kubernetes Cluster-Autoscaler

一个按需伸缩物理资源的组件

Sheduler

CA和scheduler公用了同一个库，尝试调度器的调度逻辑，来获取Pending的Pod。

Predicate：确定Pod是否可以调度到某个Node 上，基于CPU/GPU/Memory等。

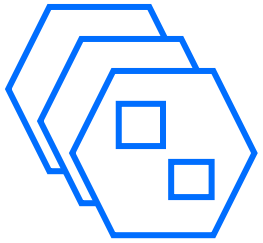
Priority：确定有一组Node满足条件，某个Node承载该Pod的优先级，考虑Deployment分摊，NodeAffinity 等。

Find Ranking：组合所有的Priority ，得出最合适的Node。

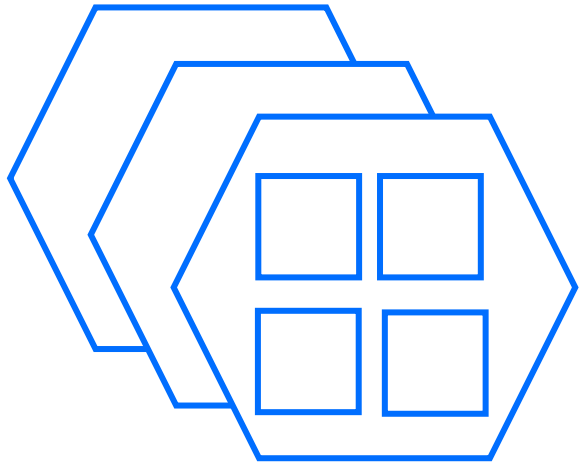
CA**复用的部分**是Predicate，这些条件大概有20个左右。

Node Group

Node Group代表了同一类型的机器的集合，每个Node Group中的机器也是CA调度节点的模板，这个Node Group可以对应到云厂商的ASG，通过云厂商的自动扩展组API，向集群内部添加和删除机器。



small node with small pods



large node with large pods

Pending Pods

CA的处理对象就是无法调度的Pending Pods，整个调度逻辑，默认每10s执行如下逻辑：

1. 检查集群中Node Group的健康状态。
2. 尝试解决集群现有的错误，如果有节点长期没有加入到集群中，尝试删除并且重试。
3. 找到无法调度的Pending Pods，如果Pods是刚创建的也会暂时过滤，防止其实可以调度这个Pod。
4. 找到一个Node Group通过扩展一定数量的机器可以让Pending Pods运行，Node的模板从公有云的API中获取，这也是为什么需要使用scheduler的逻辑。
5. 在模拟状态下，确定选出的Node Group需要扩展的节点数量。
6. 决定最佳的扩展策略（expander option），最便宜/最小/最大，执行真正的扩展动作。
7. 查找低使用率（<50%，没有无法迁移的Pod，没有无RelipcaSet的Pod）的Node。
8. 删除长时间保持上述状态的Node，默认10min。

实际上选择Node的过程是一个装箱问题，目前都是贪心策略做的近似解，并不是完全契合的扩展策略。

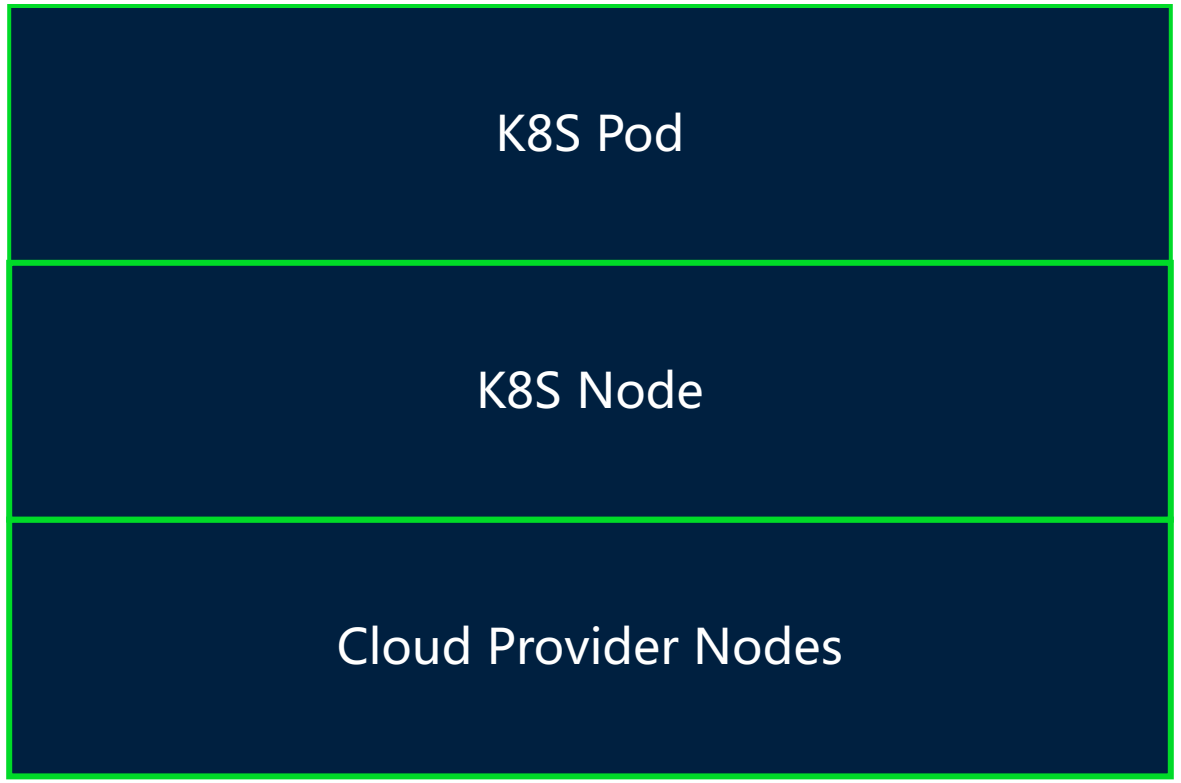
{ 2 }

预测性伸缩

基于按需扩展的集群伸缩组件进行扩展

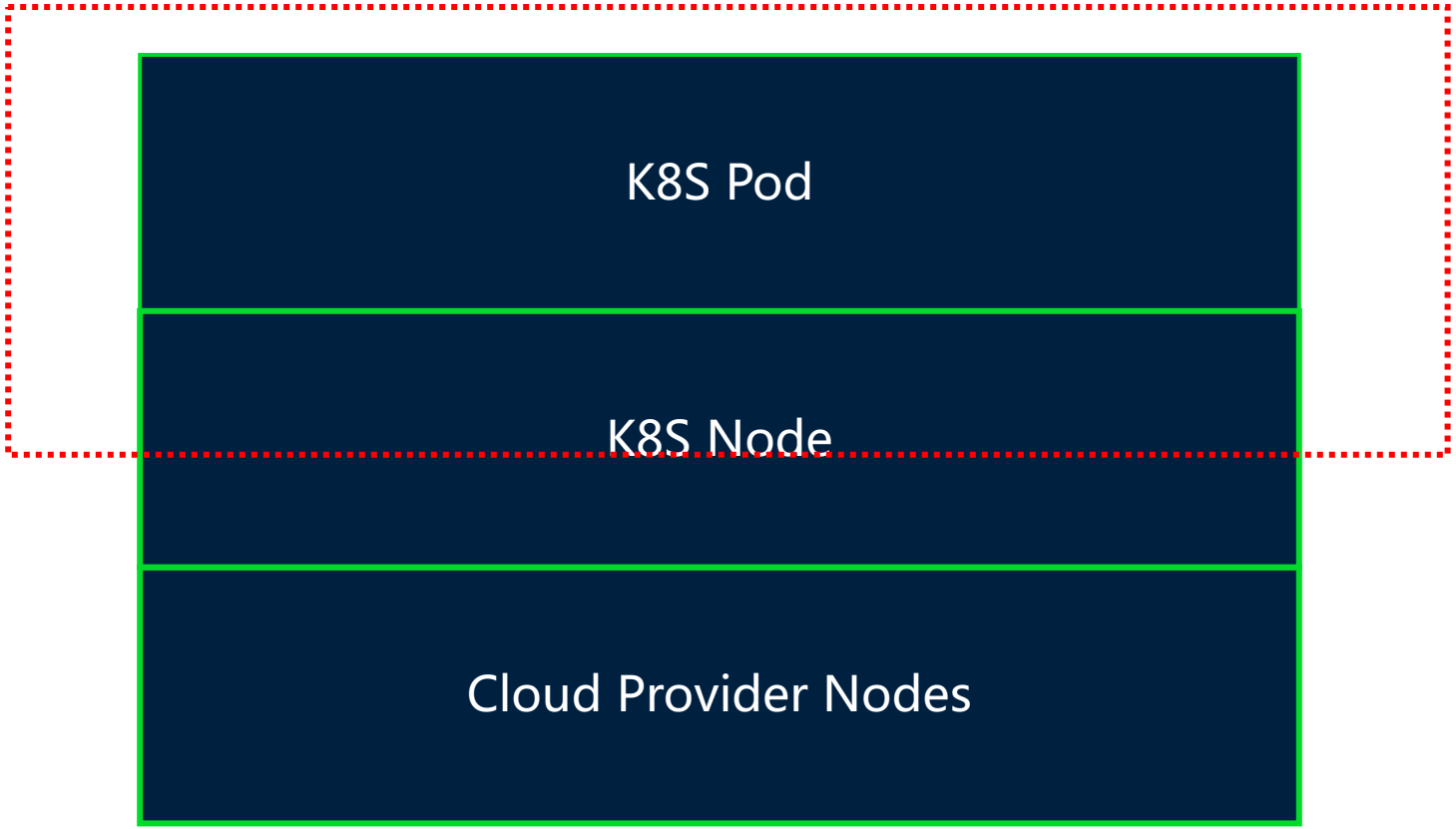
Predictive Cluster-Autoscaler

设计的分层



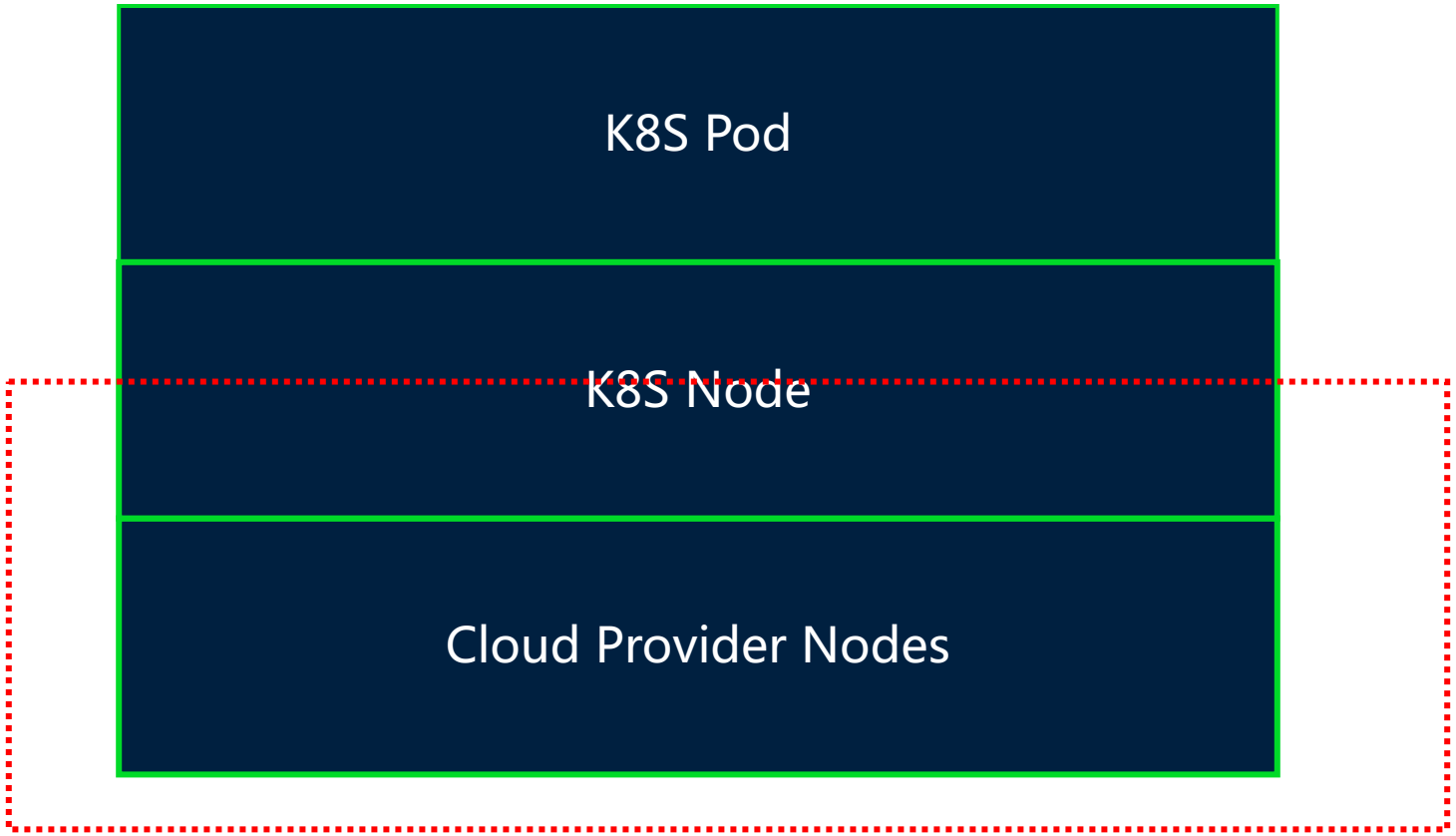
Predictive Cluster-Autoscaler

从用户的角度，资源是无限的，可以自由分配 pod



Predictive Cluster-Autoscaler

从集群的角度，按照现有的需求，以最低成本，最高效的方式提供资源



Predictive Cluster-Autoscaler

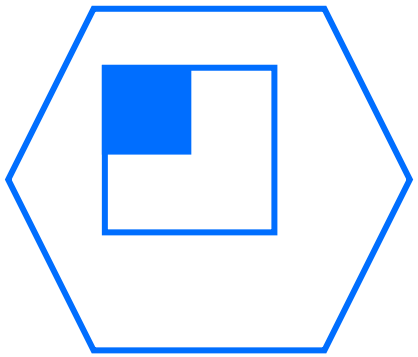
设计统一的Interface使得预测模型只需要预测接下来会需要多少Pod。
这个问题可以归类到时序预测的范畴中，需要相关人员有相应的专业背景知识，同时也要有时序预测的统计学和机器学习相关的知识。

预测算法需要统一接口，根据时间、当前 pod 的资源用量，输出需要进行扩展的Pod

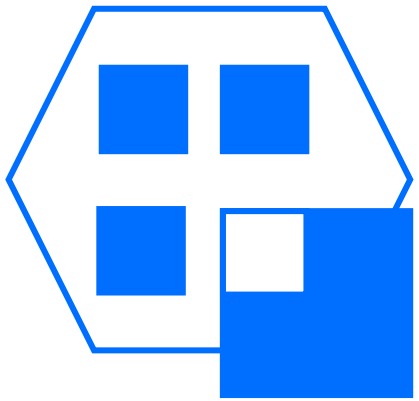
Predictive Cluster-Autoscaler

如何实现？

1 构造Pod模板，对Pod进行粒度划分（参考云厂商的Node分级），虚拟Pending Pod，将伸缩功能转接给CA。



虚拟Pod构造太大，造成浪费

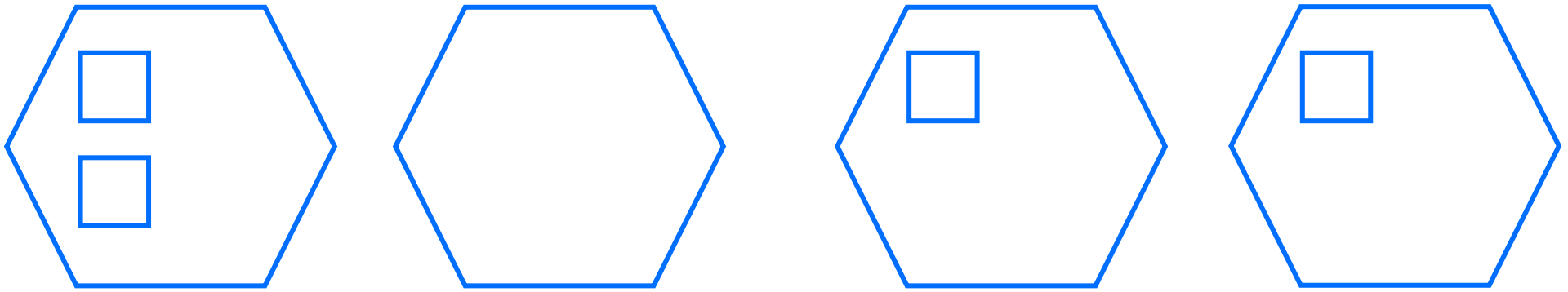


虚拟Pod构造太小，无法触发扩展

Predictive Cluster-Autoscaler

2 在完成预测性推断之后在真实的Pending Pods之后追加Predictive Pods

3 关闭一些CA的优化逻辑，在有Pending Pods的情况下始终触发Scale Down检查，并且通过贪心和时间排序把虚拟Pod尽量占领相对固定的Node。



Expected：依靠Scale Down释放Node

Unexpected：虚拟Pod正常调度，浪费资源

{ 3 }

预测算法

尝试过的一些时序预测的算法

简单的线性模型

根据当前用户资源使用情况称上一定系数作为当前需要准备的资源用量。

$$y = ax + b$$

优点：简单，不需要借助第三方的推断服务，在代码中实现静态的Interface而已，更新参数重启即可。

缺点：不够灵活，和实际情况不够符合，因为是模型线性的，在激增情况下，会放大资源的浪费。

fbprophet

算法公式

$$Y_t = S_t + T_t + R_t$$

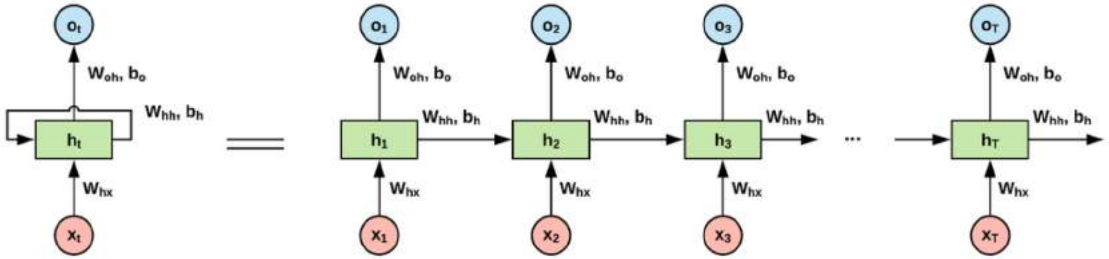
使用方式

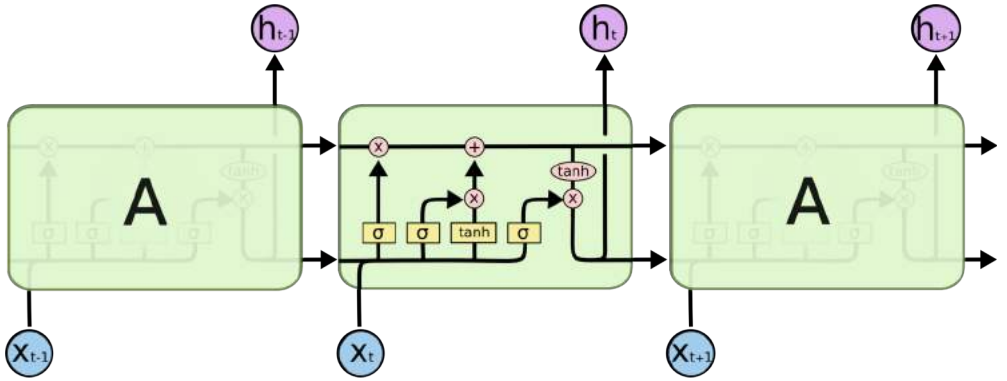
```
df = df.rename(columns={'timestamp':'ds', 'value':'y'})
m = Prophet()
m.fit(df)
future = m.make_future_dataframe(period=30s, freq=60s)
```

时序结果由季节性、趋势性和剩余项构成，支持节假日，周期规律的波动模拟，使用比较简单。
优点：使用简单，该工具对于没有相关背景的人比较友好，能够得到非常好的效果。
缺点：是一个 Python 库，只能以 Python Service 的方式部署，输入只有时间，对于突发情况无法应对。

LSTM

RNN的一种变种，输出可以作为输出，可以结合Pod的资源分配和时间进行动态的预测，相对来说更为灵活。





RNN　　LSTM

优点：输出做为输入，可以把突发情况也统一到模型当中，算法可以交由数据部门优化，后端无感。

缺点：需要独立部署模型服务，以pb或者onnx的格式部署，相对容易。

{ 4 }

总结

1. CA是一个用于在公有云上进行集群自动伸缩的组件。
2. 架构分层和专业分层要求接口的分层（后端工程师和算法工程师）。
3. 按照公有云的虚拟机分级，构造虚拟Pod。
4. 时序预测的方法，线性模型/fbprophet/LSTM 介绍。