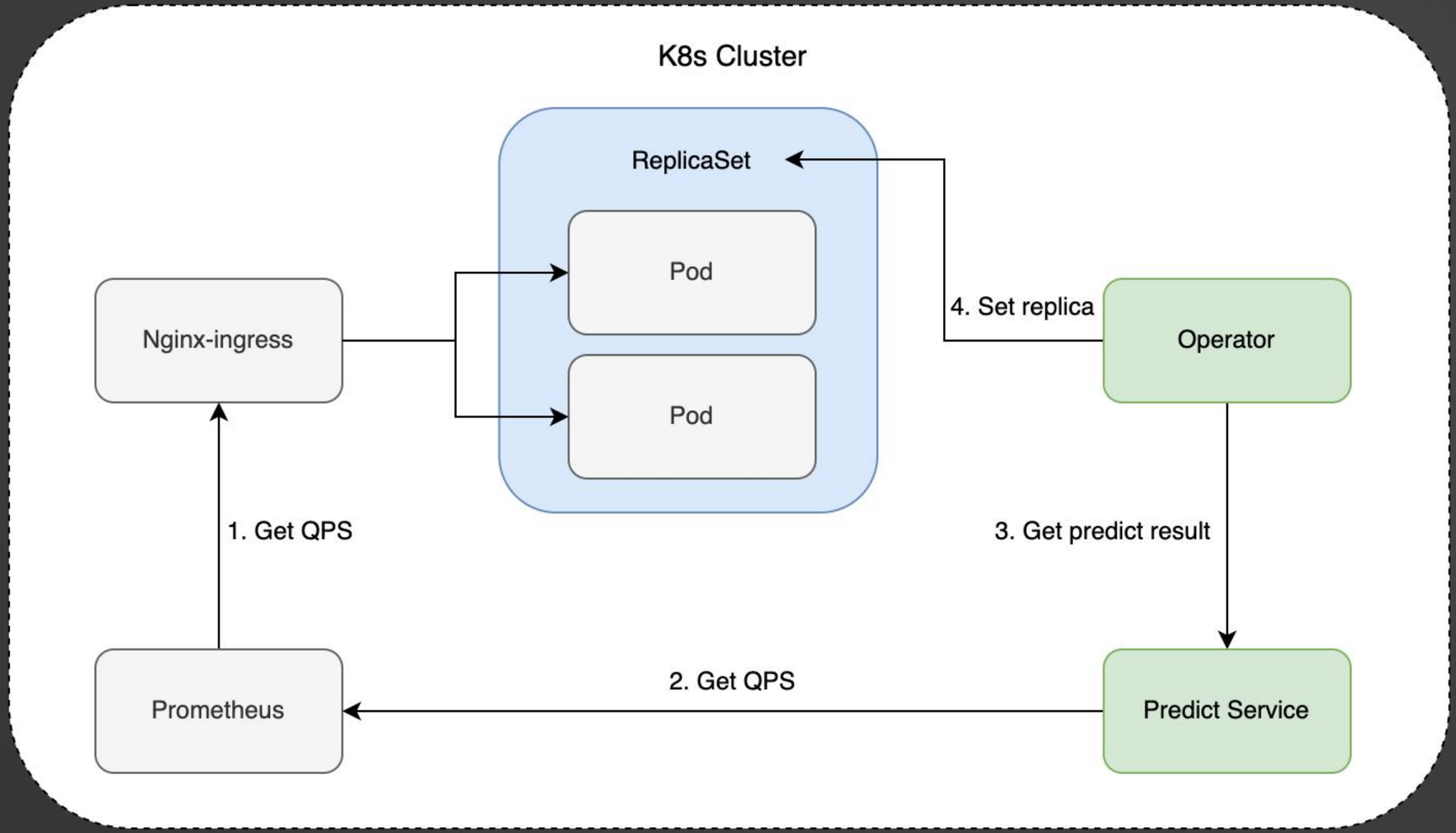
模块十： AIOps 模型训练和自动扩容

王炜/ 前腾讯云 CODING 高级架构师

目录

1. 概述

架构设计


K8s Cluster
ReplicaSet
Pod
Pod
Nginx-ingress
4. Set replica
Operator
1. Get QPS
3. Get predict result
Prometheus
2. Get QPS
Predict Service

Operator

- 从预测服务中获取推荐的副本数
- 通过修改 deployment.Spec.Replicas 字段达到控制业务 Pod 副本数的目的
- 每隔一段时间（30s）再次从预测服务中获取推荐副本数

其他与副本数存在关系的特征

特征	特征关系	备注	算法
时间	季节性	明显的波峰和波谷	季节性算法（statsmodels、Prophet）
QPS	线性	QPS 越高，副本数越多	线性回归
响应时间（延迟）	非线性关系		支持向量回归（SVR）
内存使用率	非线性		随机森林、MLP
CPU 使用率	非线性		决策树、随机森林

训练数据（模拟一天）

时间	QPS	副本数
00:00:00	10	2
00:10:00	12	3
00:20:00	15	4
00:30:00	18	5
……	……	……
23:50:00	2	1

使用的核心库： Pandas

- Pandas
  - 主要功能包括数据清洗、数据转换、数据聚合、处理缺失值、数据筛选等
- 常见操作：
  - read_csv(): 从 CSV 文件读取数据。
  - dropna(): 删除缺失值。
  - groupby(): 数据分组操作。
  - merge(): 合并数据集。

使用的核心库：Sklearn

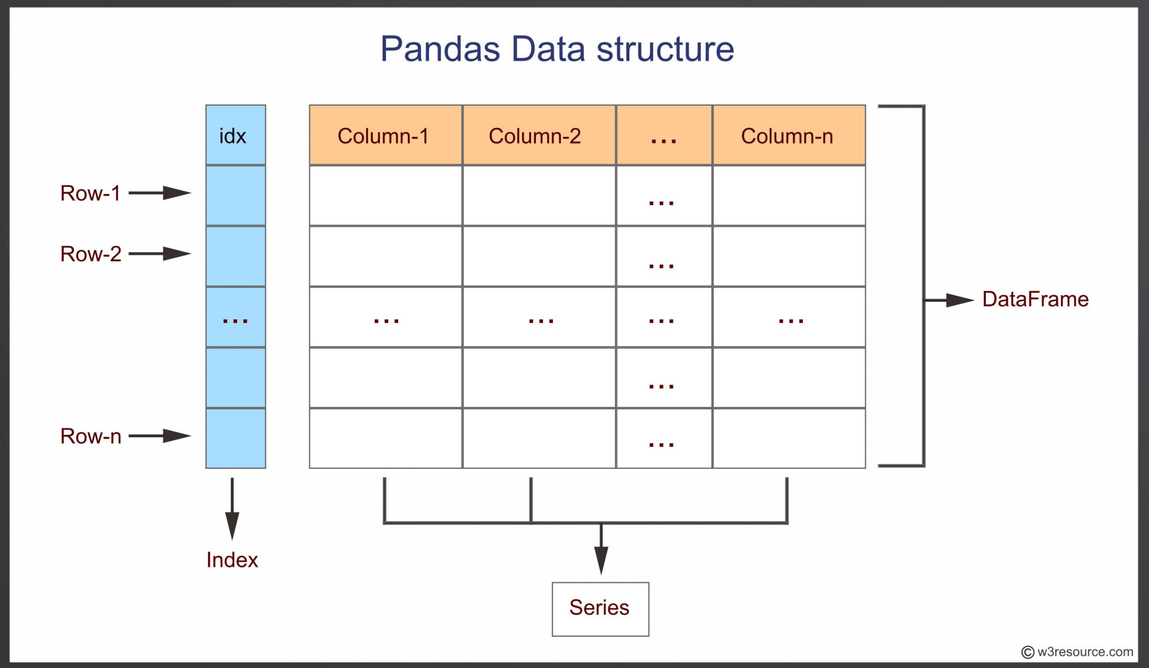
- Sklearn
  - 是一个用于机器学习的库，提供了大量的算法和工具，用于数据预处理、模型训练、模型评估和模型选择
- 常见功能：
  - 数据预处理：StandardScaler（标准化数据）、LabelEncoder（标签编码）
  - 模型训练：如 LinearRegression（线性回归）、RandomForestClassifier（随机森林分类器）
  - 模型评估：cross_val_score()（交叉验证）、accuracy_score()（准确率评估）

2. 数据处理的基本步骤

数据载入

python 复制代码

```
df = pd.read_csv("data.csv")
```



- 常见操作
  - 读取 QPS 列：qps_column = df['QPS']
  - 读取第二行：second_row = df.iloc[1]
  - 读取第二行，QPS 列的数据：second_row_qps = df.loc[1, 'QPS']
  - 筛选 QPS > 10 的行：filtered_rows = df[df['QPS'] > 10]

时间格式的转化

python 复制代码

```
df["minutes"] = (
    pd.to_datetime(df["timestamp"], format="%H:%M:%S").dt.hour * 60
    + pd.to_datetime(df["timestamp"], format="%H:%M:%S").dt.minute
)
```

- 将 timestamp 列转为时间格式，然后转换为“从午夜开始的分钟数”
  - 例如，00:10:00 会被转换为 10 分钟

捕捉时间的周期性

python 复制代码

```python
df["sin_time"] = np.sin(2 * np.pi * df["minutes"] / 1440)
df["cos_time"] = np.cos(2 * np.pi * df["minutes"] / 1440)
```

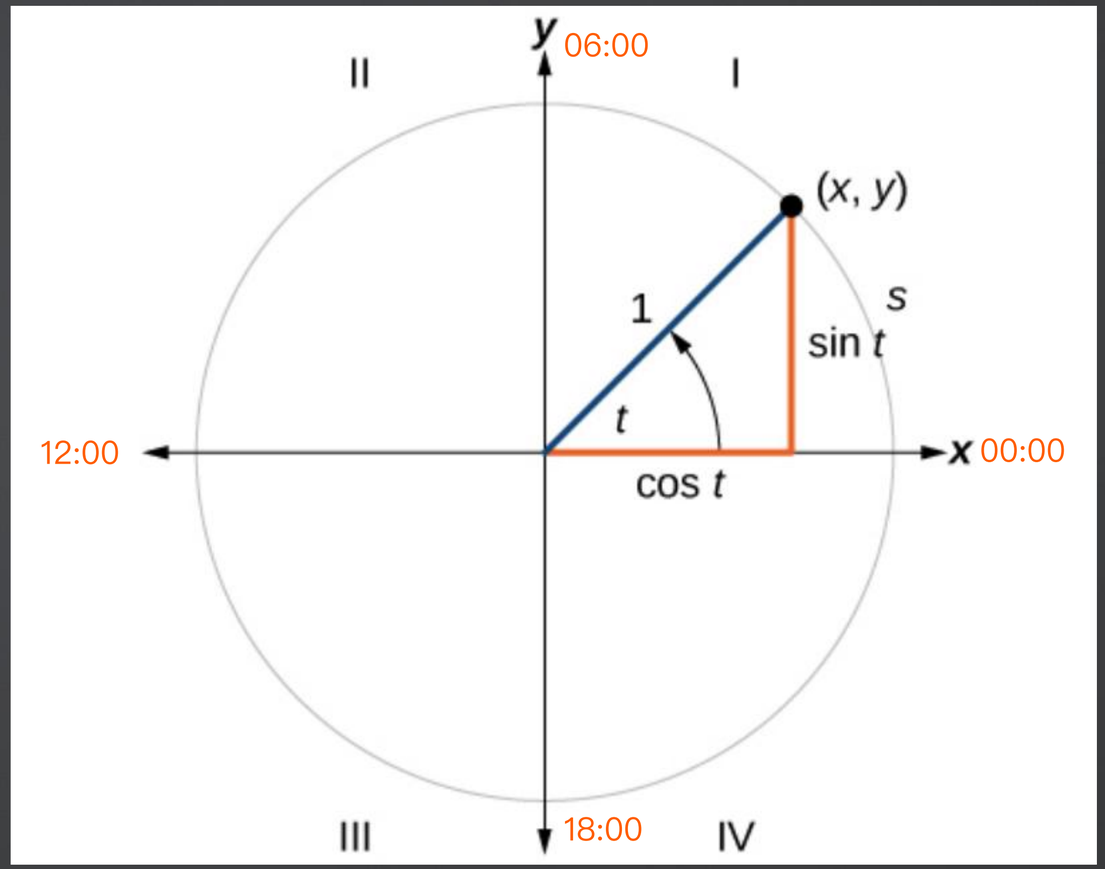
- 使用正弦和余弦函数，将时间转换为周期性特征
  - 1440 为一天的分钟数，通过周期函数让模型更好感知时间如何影响流量
- 输出：

makefile 复制代码

```makefile
sin_time: [-1.0, -0.98, ...]
cos_time: [0.0, 0.21, ...]
```

为什么要转化为余弦、正弦函数?

- 将圆周等分成 4 个象限，分别对应 0 点、上午 6 点、中午 12 点、下午 6 点
- 圆代表周期性时间循环，对于任意的时间都可以放在圆的某个点表示
- 对于任何一个圆上的点，我们可以用 cos(t)、sin(t) 表示 X 和 Y 轴的坐标
- 该坐标值就是该时间点的特征编码



其他特征的考虑

- 为了提升准确性，可以给时间增加更多特征
  - 星期几
  - 节假日
  - 月份

3. 模型训练与评估

设置特征和预测值

python 复制代码

```python
X = df[["QPS", "sin_time", "cos_time"]]
y = df["instances"]
```

- X 表示特征数据，包含：
  - QPS: 查询每秒的数量
  - sin_time: 用于捕捉时间周期性的信息
  - cos_time: 用于捕捉时间周期性的信息
- Y 表示目标变量，即我们要预测的实例数 (instances)

数据分割

split.py

```
1  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
   random_state=0)
```

- 使用 train_test_split 将数据分为训练集和测试集
- test_size=0.2 意味着 20% 的数据用于测试，80% 用于训练
- 测试集用来评估模型性能

标准化数据

StandardScaler.py

```
1  scaler = StandardScaler()
2  X_train_scaled = scaler.fit_transform(X_train)
3  X_test_scaled = scaler.transform(X_test)
```

- StandardScaler 用于将特征数据标准化，使其均值为 0，标准差为 1
- 通过 fit_transform 对训练集进行标准化，然后用 transform 处理测试集
- 标准化有助于提高模型的性能

模型评估

mean_squared_error.py

```
1  y_pred = model.coef_ * X_test_scaled + model.intercept_
2  mse = mean_squared_error(y_test, y_pred.sum(axis=1))
```

- y_pred 是模型对测试集的预测结果
- 计算均方误差 (MSE) 来评估模型的预测性能：MSE 越小，模型越好
- 评估模型可以帮助我们了解其准确性，并决定是否需要改进或调整模型

4. 模型保存与托管

保存模型

joblib.py

```
1  joblib.dump(model, "time_qps_auto_scaling_model.pkl")
```

- 使用 joblib 库将训练好的模型保存为模型文件
- 直接使用模型文件，而不需要重新训练

保存标准化器

joblib.py

```
1  joblib.dump(scaler, "time_qps_auto_scaling_scaler.pkl")
```

- 因为对数据进行了标准化，所以需要在推理时保持相同的标准
- 使用 joblib 将标准化器保存为文件
- 在使用模型进行预测时，必须确保输入数据经过相同的标准化步骤

使用模型推理

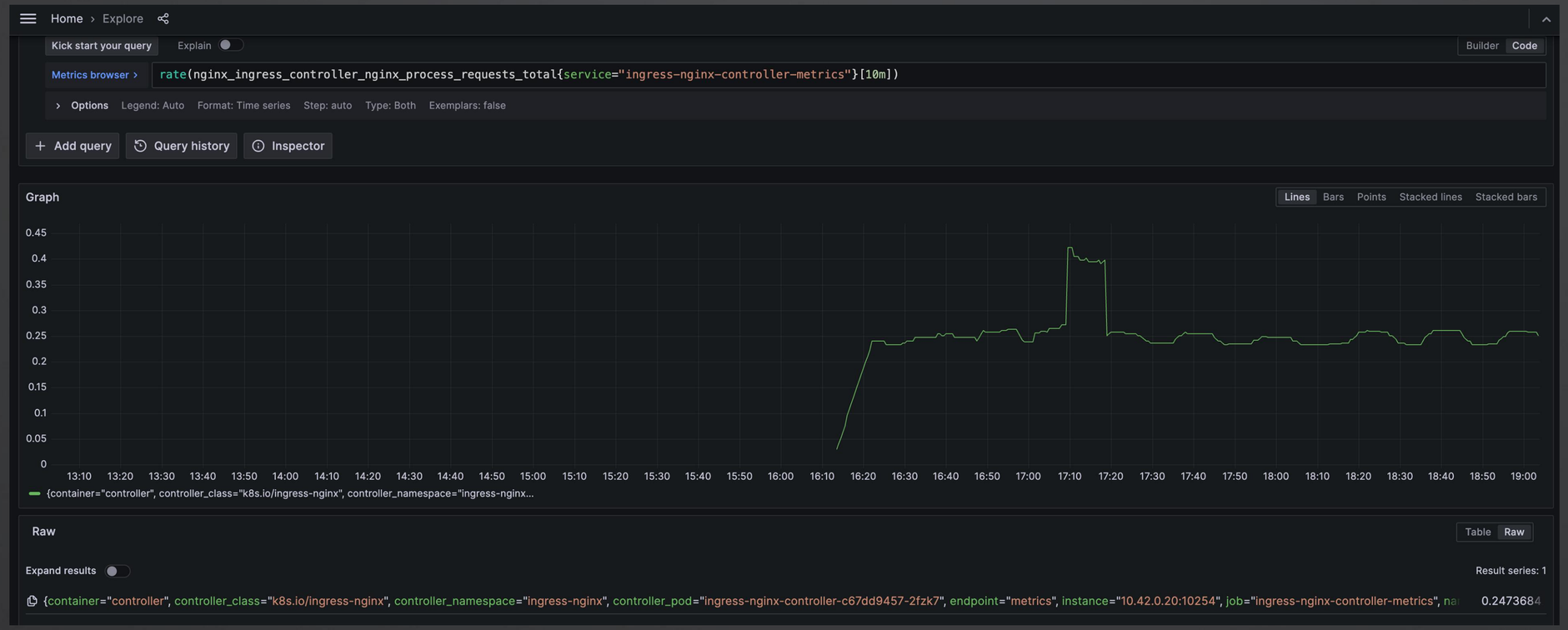
predict.py

```python
# 加载模型和标准化器
model = joblib.load("time_qps_auto_scaling_model.pkl")
scaler = joblib.load("time_qps_auto_scaling_scaler.pkl")

# 特征向量
data = {"QPS": [qps], "sin_time": [sin_time], "cos_time": [cos_time]}

df = pd.DataFrame(data)
features_scaled = scaler.transform(df)

# 预测
prediction = model.predict(features_scaled)
```

- 使用 joblib.load 加载保存的模型和标准化器，输入 QPS、时间参数
- 通过调用 predict 方法进行预测，得到实例数预测结果
- 整合 HTTP Server 提供 API 推理服务，例如 Python Flask、Golang Gin 框架等

获取 QPS 来源

- 从 Prometheus API 接口获取 Nginx-ingress 网关的 QPS
  - http://{Prometheus_host}/api/v1/query
  - rate(nginx_ingress_controller_nginx_process_requests_total{service="ingress-nginx-controller-metrics"}[10m])



打开 Nginx-ingress Metrics 开关

- 开启 Nginx-ingress Metrics
  - controller.metrics.enabled=true
  - controller.metrics.serviceMonitor.enabled=true
  - controller.metrics.serviceMonitor.additionalLabels.release=kube-prometheus-stack

helm.sh

```
helm upgrade --install ingress-nginx ingress-nginx \
--repo https://kubernetes.github.io/ingress-nginx \
--namespace ingress-nginx \
--create-namespace \
--set controller.metrics.enabled=true \
--set controller.metrics.serviceMonitor.enabled=true \
--set controller.metrics.serviceMonitor.additionalLabels.release=kube-prometheus-stack
```

将模型封装成推理服务

predict.py

```python
import ......

app = Flask(__name__)

# 定义函数从 Prometheus 获取 QPS
def get_qps_from_prometheus():
    host = "prometheus.prometheus.svc.cluster.local:9090"
    url = f"http://{host}/api/v1/query"
    query =
'rate(nginx_ingress_controller_nginx_process_requests_total{service="ingress-
nginx-controller-metrics"}[10m])'
    ......
    return qps

# 定义预测接口
@app.route("/predict", methods=["GET"])
def predict():
    try:
        qps = get_qps_from_prometheus()
        ......
        # 预测
        prediction = model.predict(features_scaled)

# 运行服务
if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=8080)

```

5. Operator 开发：获取实时预测结果并扩容

获取预测结果

- 通过请求推理服务的 /predict API 接口获取推理结果
- 得到预测实例数

predict.go

```go
func getRecommendedInstances(predictHost string) (int32, error) {
  resp, err := http.Get(fmt.Sprintf("http://%s/predict", predictHost))
  if err != nil {
    return 0, err
  }
  defer resp.Body.Close()

  body, err := io.ReadAll(resp.Body)
  if err != nil {
    return 0, err
  }

  var data struct {
    Instances int32 `json:"instances"`
  }
  if err := json.Unmarshal(body, &data); err != nil {
    return 0, err
  }
  return data.Instances, nil
}
```

弹性扩容/缩容

- 调用 r.Update(ctx, deployment) 方法更新 deployment.Spec.Replicas 字段
- 实现弹性扩容/缩容

update.go

```go
if *deployment.Spec.Replicas != recommendedReplicas {
  deployment.Spec.Replicas = &recommendedReplicas
  err := retry.RetryOnConflict(retry.DefaultRetry, func() error {
    return r.Update(ctx, deployment)
  })
  if err != nil {
    logger.Error(err, "failed to update Deployment replicas")
    return ctrl.Result{}, err
  }
  logger.Info("Updated Deployment replicas", "replicas", recommendedReplicas)
}
```

步骤

- mkdir hpa-operator && cd hpa-operator
- go mod init github.com/lyzhang1999/hpa-operator
- kubebuilder init --domain=aiops.com
- kubebuilder create api --group hpa --version v1 --kind PredictHPA
- 修改：api/v1/predicthpa_types.go PredictHPASpec
- 修改：internal/controller/predicthpa_controller.go 增加相关业务逻辑
- 创建 sample/hpa_v1_predicthpa_local.yaml 文件
- 本地测试转发推理服务到本地
- 如果将 Operator 部署到集群
  - predictHost: "machine-learning-python:8080"

课后作业

- 思考如何提升模型预测的准确度，给出实施的方案和必要的代码

THANKS