# Practical Efficient Microservice Autoscaling with QoS Assurance

## Methodology

* Instead of finding the best resource configuration, it first allocates abundant resources to all microservices to satisfy SLO and tries to exploit resource reduction opportunities.
* Use the monotonic resource reduction. Reduce it or do not change it.
* Use the response time as feedback to identify resource reduction opportunities
* Experiments: three prototypes: TrainTicket, SockShop and DeathStarBench.
* Use Prometheus to collect container-specific metrics such as CPU utilization and CPU throttling.
* Use Linkerd to collect end-to-end latency performance.
* Use Jaeger to collect traces.

It is open source. [GitHub](https://github.com/rajibhossen/microservice-autoscaling)

## The Core Problem

For efficient microservice management, it is crucial to identify how resources should be distributed among different microservices as the same amount of resources can result in significantly different performance based on which microservice gets how much resources. However, finding the efficient resource distribution is very hard as there are no easily generalizable markers to assist in the resource allocation.

## Design

* Assure QoS
* find efficient resource allocation

Use a discrete-time model with a time step

Define the following things:

### Design Principles

* Learning with a feedback.
* Do not find the best, find the better with few iterations.
* QoS preserving learning.
* Feedback-based navigation.

### PEMA

Update resource allocation in regular intervals based on the response time observed in the previous interval. Time interval is 2 minutes. For resource reduction at time step , we first decide the number of microservices to reduce resources from using

* : response time in the previous time step.
* : user-defined non-negative parameter that determines how aggressively we want to reduce the source.

Next decide how much resource we reduce the resource in the microservice

Avoid bottleneck services. For the -th microservice, we denote its utilization as with a bottleneck threshold and CPU throttling time as with a bottleneck .

Find the index:

Normalize the utilization:

Update the probability: