Scaling of containerized network functions

Mohanad Elamin
University of Amsterdam
melamin@os3.nl

Pim Paardekooper
University of Amsterdam
ppaardekooper@os3.nl

Supervisor:
Jamila Alsayed Kassem
UvA
Introduction

- EPI framework
  - Secure connection
  - Setup bridging function
- Scaling bridging functions
  - Horizontal
  - Vertical
What is the impact of varying k8s autoscaling thresholds for bridging functions on end-user application traffic?

- What metrics should trigger the scaling?
- What is the impact of the bridging functions horizontal scaling on application traffic?
- What is the impact of reconfiguration on in-transit traffic?
Related Work

- Duc-Hung et al. explained the autoscaling mechanisms available in existing container orchestrators in the IT domain, with special focus on the mobile core elements.

- Salman Taherizadeh and Marko Grobelnik proposed three key factors which should be considered for auto-scaling methods in Kubernetes.

- Steven Van Rossem et al. points out that predicting the performance of a VNF chain based on the performance of the discrete network functions is not accurate.

- Adel Nadijaran Toosi et al. research produces a tool called ElasticSFC, which shows that their auto-scaling techniques based on the VNF chain can reduce cost.
Methodology

01 PoC
Build Proof of Concept on top of the EPI framework

02 Identification
Identify scaling possibilities

03 Experiment
Define and do experiments based on the scaling possibilities

04 Plotting
Create graphs that will answer our research question

05 Conclusion
Answer the research question based on the plots
Setup - Topology

1. Testing machine
2. Locust load balancer
3. Socks5 proxy
4. Bridging function
5. Httpbin server
Setup - Locust Load Generator

- Load tests:
  - Number of users
  - Spawn rate
  - Tasks

- Locust metrics:
  - RPS
  - Users
  - Response time
Kubernetes autoscalers implementations

- **HPA**
  - desiredReplicas = ceil[currentReplicas * (currentMetricValue/desiredMetricValue)]

- **VPA**
  - Desired value calculation
    - Recommender
    - Updater
    - Admission plugin

- **Metric server**
- **Custom metrics**
  - Prometheus
Experiments

- No VPA
- Increasing HPA threshold for CPU utilization
- Increasing number of users in locust load generator
- Data collected:
  - Response time
  - CPU usage
    - $1 \text{ CPU} = 1 \text{s} = 1000 \text{ milicore} = 1 \, 000 \, 000 \, 000 \, \text{n}$
    - CPU in nanoseconds/1 000 000 = CPU in milicore
## Test Scenarios

<table>
<thead>
<tr>
<th>Test #</th>
<th># of Users</th>
<th>Spawn rate</th>
<th>Run time (Sec)</th>
<th>HPA MAX REPLICA</th>
<th>HPA Utilization</th>
<th>CPU Limit</th>
<th>Memory Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>100</td>
<td>5</td>
<td>30</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1</td>
<td>100</td>
<td>5</td>
<td>50</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>1</td>
<td>100</td>
<td>5</td>
<td>80</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>1</td>
<td>100</td>
<td>5</td>
<td>90</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>10</td>
<td>100</td>
<td>5</td>
<td>30</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>10</td>
<td>100</td>
<td>5</td>
<td>50</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>10</td>
<td>100</td>
<td>5</td>
<td>80</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>10</td>
<td>100</td>
<td>5</td>
<td>90</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>9</td>
<td>1000</td>
<td>100</td>
<td>100</td>
<td>5</td>
<td>30</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>100</td>
<td>100</td>
<td>5</td>
<td>50</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>11</td>
<td>1000</td>
<td>100</td>
<td>100</td>
<td>5</td>
<td>80</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>12</td>
<td>1000</td>
<td>100</td>
<td>100</td>
<td>5</td>
<td>90</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>13</td>
<td>10000</td>
<td>1000</td>
<td>100</td>
<td>5</td>
<td>30</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>14</td>
<td>10000</td>
<td>1000</td>
<td>100</td>
<td>5</td>
<td>50</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>15</td>
<td>10000</td>
<td>1000</td>
<td>100</td>
<td>5</td>
<td>80</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>1000</td>
<td>100</td>
<td>5</td>
<td>90</td>
<td>1000m</td>
<td>500Mi</td>
</tr>
</tbody>
</table>
Setup - Automated Deployment and Testing

1. Load Test use case
2. Test Automation Script
3. Deploy Setup
4. Trigger Load Test
5. Metrics gathering
6. Collect Data
7. Plot Data

Test Use Cases
Helm Charts
Locust Load Generator
HPA
BFs
Proxy
Server
Data Collector Script
Plotting
CSV
By default, Locust uses an HTTP persistent connection (keep-alive).

Kubernetes service pin the session to single Bridging Function.

The load is not distributed after scale-out events.

‘Connection’: 'close' can be used to change the behavior.

Vertical scaling is a better fit for such traffic.
Results - Horizontal Scaling: CPU

- Bridging function CPU is more balanced with short lived sessions.
- The HPA operates on a ratio between desired and current metrics.
- The scale out event is more aggressive if the threshold is low.
  - Bursty traffic

![Graph showing utilization and CPU usage over time, with initial BF CPU and outscaling events marked.](image)

- HPA Threshold = 30%
- Load is distributed
Results - Horizontal Scaling: Memory

- CPU is the primary metric used during testing.
- The tested bridging function consumed steady memory.
- Memory usage not decreasing after scaling out.
Results - Horizontal Scaling: Response Time

- The Locust load generator records the response time of the HTTP requests.
- Response time increases after the autoscaling events due to the load-balancing overhead.
- Bridging function high CPU is not causing an increase in response time.
- The processing logic of the bridging function used is very insignificant.
Conclusion

- The metric for triggering Autoscaling is highly dependant on the bridging function logic.
  - CPU.
  - Memory.
  - Custom Metrics.
- Long lived sessions doesn’t benefit from Horizontal Scaling.
- There is no single optimum threshold to trigger autoscaling.
- Overhead added by autoscaling can impact application traffic.
Future Work

● Use data analysis and machine learning to find the best threshold for different metrics and traffic use cases.
● Test with different application traffic.
● Use production grade bridging function.
● Research the impact of bridging function chaining.
● Vertical Pod Autoscaling.
Thank You

https://github.com/mohanadelamin/rp2-epif

Mohanad Elamin  
University of Amsterdam  
melamin@os3.nl

Pim Paardekooper  
University of Amsterdam  
ppaardekooper@os3.nl

Supervisor:  
Jamila Alsayed Kassem  
UvA