Virtualization, Elasticity and Performance for Distributed Applications

Hong-Linh Truong
Faculty of Informatics, TU Wien

hong-linh.truong@tuwien.ac.at
www.infosys.tuwien.ac.at/staff/truong
@linhsolar
What this lecture is about?

- Resources and their impact on distributed systems and applications
- Virtualization
  - Resource virtualization
- Elasticity
  - Key concepts and techniques
- Performance
  - Utilizing virtualization and elasticity for some performance patterns
Impact of resources on Distributed applications

Types of distributed applications

- Workflow/process style
- Data-centric pipeline style

Some questions for DevOps

- How to have a development environment that is similar to the operational/production one?
- How to utilize computing resources in the best way?
- How to achieve the best performance?

Figure sources: http://www.cloudcomputingpatterns.org/Distributed_Application
Recall – Breakdown the complexity

Figure source: Sam Newman, Building Microservices, 2015

How to make sure that the underlying resources and infrastructures are suitable for “small autonomous services”?
Concepts of today’s lecture

Virtualization

Elasticity

Performance
VIRTUALIZATION
What is virtualization? A bird view

- Virtualization:
  - To abstract low-level compute, data and network resources to create *virtual version* of these resources

- Virtualization software creates and manages “virtual resources” isolated from physical resources

→ Virtualization is a powerful concept: we can apply virtualization techniques virtually for everything!

→ Virtualization is a key enabling technology for cloud computing and modern computer networks.
Virtualizing physical resources

Web Services (e.g. REST)
Service Container (e.g. Tomcat)
Operating System (e.g. Ubuntu)
Physical resources (e.g. 4 dual-core CPUs +8GB RAM)

Web Services (e.g. REST)
Service Container (e.g. Tomcat)
Operating System (e.g. Windows)
Virtualization layer/resources (e.g., 1 dual-core CPU, 2 GB RAM)
Operating System (e.g. Ubuntu)
Physical resources (e.g. 4-core CPU +8GB RAM)

So if we just develop „Web services“, why is it important to us?
Main types of virtualization of infrastructures for distributed apps

- **Compute resource virtualization**
  - Compute resources: CPU, memory, I/O, etc.
  - To provide virtual resources for "virtual machines"

- **Storage virtualization**
  - Resources: storage devices, harddisk, etc.
  - To optimize the usage and management of data storage

- **Network Function Virtualization**
  - Network resources: network equipment & functions
  - To consolidate network equipment and dynamically provision and manage network functions
Compute Resource Virtualization Technologies

- Physical compute resources:
  - Individual physical hosts/servers (CPU, memory, I/O)
  - Clusters and data centers
- At the low-level: two main streams
  - Hypervisor/Virtual Machine monitor
    - Virtual machines (VirtualBox, VMWare, Zen, etc.)
  - Containerization
    - Containers (Linux Containers, Docker, Warden Container, OpenVZ, OCI based containers, etc.)
Hypervisor/Virtual Machine Monitor

Virtual machine

- App
- App
- App

- Guest OS
- Virtual Machine Monitor/Hypervisor (Level 2)
- Host OS
- Physical resources

Another model (Hypervisor level 1)

Xen Project Hypervisor

- Open Source
- Storage
- Control Interface
- Drivers

Xen Control Interface

Hypervisor

Virtualized Hardware

Hardware

Network and Storage I/O

Linux

RAM/CPU

Windows

https://www.citrix.de/products/xenserver/tech-info.html
Containers

https://www.docker.com/what-docker

http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6903537
We do not dig into low-level techniques in virtualization, but examine

- How would virtualization techniques enable us to acquire, utilize and manage resources for our Devs and Ops of distributed applications and systems?

- How would such techniques change our software design?

- How to align on-demand resources/infrastructures with software using them
## Virtual machines versus containers

### Table 1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Virtual Machines</th>
<th>Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest OS</td>
<td>Each VM runs on virtual hardware and Kernel is loaded into its own memory region</td>
<td>All the guests share same OS and Kernel. Kernel image is loaded into the physical memory</td>
</tr>
<tr>
<td>Communication</td>
<td>Will be through Ethernet Devices</td>
<td>Standard IPC mechanisms like Signals, pipes, sockets etc.</td>
</tr>
<tr>
<td>Security</td>
<td>Depends on the implementation of Hypervisor</td>
<td>Mandatory access control can be leveraged</td>
</tr>
<tr>
<td>Performance</td>
<td>Virtual Machines suffer from a small overhead as the Machine instructions are translated from Guest to Host OS.</td>
<td>Containers provide near native performance as compared to the underlying Host OS.</td>
</tr>
<tr>
<td>Isolation</td>
<td>Sharing libraries, files etc between guests and between guests hosts not possible.</td>
<td>Subdirectories can be transparently mounted and can be shared.</td>
</tr>
<tr>
<td>Startup time</td>
<td>VMs take a few mins to boot up</td>
<td>Containers can be booted up in a few secs as compared to VMs.</td>
</tr>
<tr>
<td>Storage</td>
<td>VMs take much more storage as the whole OS kernel and its associated programs have to be installed and run</td>
<td>Containers take lower amount of storage as the base OS is shared</td>
</tr>
</tbody>
</table>

VM versus containers

Fig. 6. Random I/O throughput (IOPS).

Fig. 11. MySQL throughput (transactions/s) vs. CPU utilization.

Fig. 12. MySQL latency (in ms) vs. concurrency.

Source: Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio:
An updated performance comparison of virtual machines and Linux containers. ISPASS 2015: 171-172
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7095802
Examples of performance

Fig. 8. Evaluation of NoSQL Redis performance (requests/s) on multiple deployment scenarios. Each data point is the arithmetic mean obtained from 10 runs.

Fig. 10. MySQL throughput (transactions/s) vs. concurrency.

Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio: An updated performance comparison of virtual machines and Linux containers. ISPASS 2015: 171-172
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7095802
Tools, frameworks and providers: Chef, Vagrant, Amazon, Google, Microsoft, OpenStack, …
Interactions in VMs/containers provisioning and management

You focus on application development, how does it impact your work?

Google Container Registry
AWS EC2 Container Registry
Azure Container Registry

Registry

VM/container images
configuration artifacts

build

deploy
backup/store

code

Runtime Management

Virtual machine/container instances

Operating System
Physical resources
Examples

Source: https://docs.docker.com/engine/understanding-docker/
IBM Cloud OpenStack Services runs on OpenStack Icehouse to provide you with an environment built on the most current open standards.

Other OpenStack components included:
- Heat – for pattern orchestration
- Ceilometer – for reporting, metering

OpenStack experimental projects are not enabled by default:
- Trove
- Sahara

*Optional

Source: http://www.slideshare.net/OpenStack_Online/ibm-cloud-open-stack-services
Virtual data centers

- On-demand virtual data centers
  - Compute nodes, storage, communication, etc.
  - Virtual data centers work like a single distributed system (e.g., a cluster)

- Challenges
  - Provision resources/nodes (using VMs or containers)
  - Configure networks within virtual data centers
  - Configure networks between virtual data centers and the outside systems
  - Deploy software into the virtual data centers
Example - Weave Net and docker

- Work with Kubernetes & Mesos as well
- Key idea: using network plug-in for containers + P2P overlay of routers in the host

Source: https://www.weave.works/docs/net/latest/introducing-weave/
Kubernetes

- Support Docker, rkt, runc, etc.

Source: https://kubernetes.io/docs/concepts/architecture/cloud-controller/
Example -- DC/OS

Source: https://docs.mesosphere.com/1.11/overview/architecture/components/
Storage Virtualization

- Low-level storage
  - e.g., RAID (redundant array of independent disks)
- High-level, e.g., database
  - MySQL Cluster + auto-sharding

- Why is it relevant to you?
- What changes should we make in our apps?

Source:
Network Function Virtualization

- Consolidate network equipment and services
- On-demand provisioning of network functions

Is it the sysadmin task? I never see the network part in my apps. So why is it relevant to the software developer?

Figure source: https://portal.etsi.org/NFV/NFV_White_Paper.pdf
Why is resource virtualization interesting for distributed applications?

What are impacts of virtualization on the development and operation of distributed applications?
List of why and impact

- Server consolidation
  - Consolidating compute capabilities
- Security, fault tolerance and performance
  - Through dynamic provisioning and auto-scaling
- Cost/optimization
  - elasticity, hot deployment, etc.
- Compatibility issues
- DevOps
  - Closing the gap between real and development environments
Server Consolidation

- Cost, complexity (management)
  - Infrastructures (cooling, spaces), human resources
- Resources under utilization
How does it help me? Consolidation looks good for the sysadmin but not relevant to the software developer? What changes the developer has to do?
Microservices + partitioning

- Partition complex code into different services → easy configuration and maintenance
- But this has to be in sync with underlying resources provisioning (e.g., containers)
Questions from practices

How big is a microservice?
How big is a microservice?

**eShopOnContainers reference application**
(Development environment architecture)

**Client apps**
- **eShop mobile app**
  - Xamarin.Forms
  - C#
  - xPlat. OS:
  - iOS
  - Android
  - Windows
- **eShop traditional Web app**
  - HTML
- **eShop SPA Web app**
  - TypeScript/Angular 2

**Docker Host**

- **Identity microservice** (STS+users)
  - SQL Server database
- **Catalog microservice**
  - SQL Server database
- **Ordering microservice**
  - SQL Server database
- **Basket microservice**
  - Redis cache
- **Marketing microservice**
  - MongoDB / CosmosDB
  - SQL Server
- **Locations microservice**
  - MongoDB / CosmosDB

**Event Bus**
(Publish/Subscribe Channel)

Figure source: https://blogs.msdn.microsoft.com/dotnet/2017/08/02/microservices-and-docker-containers-architecture-patterns-and-development-guidance/
How many containers are needed for a microservice?
Look at some patterns

- Single container in single node
- Multi-containers in single node
- Multi-containers in multiple nodes

Sources:

https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns
Security improvement

(Virtual) server and service isolation
Fault tolerance and performance

How does resource virtualization help improving fault tolerance and performance?

- Possible benefits
  - Failure masking
  - Cost/optimization
    - Elasticity, hot deployment, etc.
    - Cloud bursting (combining private + public resources)
  - Improving service performance in incident management
    - E.g., spend time to fix a machine or just quickly relaunch a new one (and fix the old one later)?
Development and deployment

- Compatibility and support legacy application
- Maintenance
- Close the gap between development/test environment and real/production environments
- Simplify testing, emulating real environments, etc.
Service Discovery in the container age

Example of a simple service

```javascript
var express = require('express');
var app = express();
app.get('/', function(request, response) {
  response.send('Hello World! I am DST 2018');
});
var port = process.env.PORT || 3000;
app.listen(port, function() {
  console.log("Listening on ": port);
});
```

Multiple instances in VMs/containers

```
{
  "service": {
    "name": "DSTProvider",
    "tags": ["nodejs"],
    "address": "35.190.161.XXX",
    "port": 3000
  }
}
```

Why do we need the service discovery? How do you do the service discovery?
Distributed Coordination

- A lot of algorithms, etc.
  - Paxos family
- Well-known in the cloud
  - Zookeeper

Notes from the paper: “server replication (SR), log replication (LR), synchronization service (SS), barrier orchestration (BO), service discovery (SD), group membership (GM), leader election (LE), metadata management (MM) and distributed queues (Q)”

ZooKeeper Service

Source: https://zookeeper.apache.org/doc/r3.4.10/zookeeperOver.html
ZooKeeper data -- znodes

- Data nodes called znodes
- Missing data in a znode → Problems with the entity that the znode represents
- Persistent znode
  - /path deleted only through a delete call
- Ephemeral znode, deleted when
  - The client created it crashed
  - Session expired

Source:
https://zookeeper.apache.org/doc/r3.4.10/zookeeperOver.html
Consul

- https://www.consul.io
- Cross data centers
- End-to-end service discovery

Figure source: https://www.consul.io/docs/internals/architecture.html
ETCD

- Distributed key-value store

Technical Overview

etcd is written in Go which has excellent cross-platform support, small binaries and a great community behind it. Communication between etcd machines is handled via the Raft consensus algorithm.

Latency from the etcd leader is the most important metric to track and the built-in dashboard has a view dedicated to this. In our testing, severe latency will introduce instability within the cluster because Raft is only as fast as the slowest machine in the majority. You can mitigate this issue by properly tuning the cluster. etcd has been pre-tuned on cloud providers with highly variable networks.

More Information

Presentation: How Raft Works

Figure source: https://coreos.com/etcd
Some comparison

- From etcd view

<table>
<thead>
<tr>
<th></th>
<th><strong>ETCD</strong></th>
<th><strong>ZOOKEEPER</strong></th>
<th><strong>CONSUL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency Primitives</td>
<td>Lock RPCs, Election RPCs, command line locks, command line elections, recipes in go</td>
<td>External curator recipes in Java</td>
<td>Native lock API</td>
</tr>
<tr>
<td>Linearizable Reads</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-version</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Concurrency Control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>Field compares, Read, Write</td>
<td>Version checks, Write</td>
<td>Field compare, Lock, Read, Write</td>
</tr>
<tr>
<td>Change Notification</td>
<td>Historical and current key intervals</td>
<td>Current keys and directories</td>
<td>Current keys and prefixes</td>
</tr>
<tr>
<td>User permissions</td>
<td>Role based</td>
<td>ACLs</td>
<td>ACLs</td>
</tr>
<tr>
<td>HTTP/JSON API</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Membership</td>
<td>Yes</td>
<td>&gt;3.5.0</td>
<td></td>
</tr>
<tr>
<td>Reconfiguration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum reliable</td>
<td>Several gigabytes</td>
<td>Hundreds of megabytes (sometimes several gigabytes)</td>
<td>Hundreds of MBs</td>
</tr>
<tr>
<td>database size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum read</td>
<td>Network RTT</td>
<td>No read linearization</td>
<td>RTT + fsync</td>
</tr>
<tr>
<td>linearization latency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: https://coreos.com/etcd/docs/latest/learning/why.html
Elasticity in physics

“elasticity (or stretchiness) is the physical property of a material that returns to its original shape after the stress (e.g. external forces) that made it deform or distort is removed” – http://en.wikipedia.org/wiki/Elasticity_(physics)

- It is related to the form (the structure) of something
  - “Stress” causes the elasticity (structure deformation)
  - “Strain” measures what has been changed (amount of deformation)

- In the context of computing: given a process or a system
  - What can be used to represent “Stress” and “Strain”?
  - When does a “strain” signals a “dangerous situation”?
  - How to be elastic under dynamic “stress”? 

DST 2018
“Elastic computing is the use of computer resources which vary dynamically to meet a variable workload” –

“Clustering elasticity is the ease of adding or removing nodes from the distributed data store” –

“What elasticity means to cloud users is that they should **design their applications to scale their resource requirements up and down** whenever possible.“, David Chiu –
http://xrds.acm.org/article.cfm?aid=1734162
Elasticity in (big) data analytics

- More data → more compute resources (e.g. more VMs)
- More types of data → more activities → more analytics processes
- Change quality of analytics
  - Change quality of data
  - Change response time
  - Change cost
  - Change types of result (form of the data output, e.g. tree, table, story)
Diverse types of elasticity requirements

- **Application user**: “If the cost is greater than 800 Euro, there should be a scale-in action for keeping costs in acceptable limits”

- **Software service provider**: “Response time should be less than amount X varying with the number of users.”

- **Cloud infrastructure provider**: “When availability is higher than 99% for a period of time, and the cost is the same as for availability 80%, the cost should increase with 10%.”

Solving conflicting requirements across layers is challenging
General software design concept: Lifecycle of applications and elasticity

Elasticity specification

Control processes

Orchestrate concrete operations

Elasticity specification

Deployment process

Elasticity Prediction Function

Elasticity Adjustment Function

Elasticity Primitive Operations

Cloud-specific Management Function specific APIs

Requirement trigger
Process control
Behavior change

Monitoring information

Operation Time

Check: https://doi.org/10.1016/j.procs.2016.08.276
Our focus in this course: elasticity of compute resources for distributed applications

Q1: Where can elasticity play a role in these application models?

Q2: How does virtualization help implementing elasticity of resources
Practical elasticity implementation

- Elasticity specification
  - Constraints/Rules
- Elasticity monitoring and prediction
  - Can you name some monitoring techniques?
- Elasticity controller/adjustment:
  - Interpret constraints and monitoring data
  - Control
    - Reactive scale versus proactive scale
    - Vertical scaling (scale up/down) versus Horizontal scaling (scale out/in)
## Elasticity constraints

Table 1. Summary of the reviewed literature about threshold-based rules

<table>
<thead>
<tr>
<th>Ref</th>
<th>Auto-scaling Techniques</th>
<th>H/V</th>
<th>R/P</th>
<th>Metric</th>
<th>Monitoring</th>
<th>SLA</th>
<th>Workloads</th>
<th>Experimental Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>[65]</td>
<td>Rules</td>
<td>Both</td>
<td>R</td>
<td>CPU, memory, I/O</td>
<td>Custom tool, 1 minute</td>
<td>Response time</td>
<td>Synthetic, Browsing and ordering behavior of customers.</td>
<td>Custom testbed (called UC Cloud) + TPC</td>
</tr>
<tr>
<td>[72]</td>
<td>Rules</td>
<td>H</td>
<td>R</td>
<td>Average waiting time in queue, CPU load</td>
<td>Custom tool.</td>
<td>—</td>
<td>Synthetic</td>
<td>Public cloud, Future Grid, Eucalyptus India cluster</td>
</tr>
<tr>
<td>[64]</td>
<td>Rules</td>
<td>Both</td>
<td>R</td>
<td>CPU load, response time, network link load, jitter and delay.</td>
<td>—</td>
<td>—</td>
<td>Only algorithm is described, no experimentation is carried out.</td>
<td></td>
</tr>
<tr>
<td>[48]</td>
<td>Rules + QT</td>
<td>H</td>
<td>P</td>
<td>Request rate</td>
<td>Amazon CloudWatch, 1-5 minutes</td>
<td>Response time</td>
<td>Real, Wikipedia traces</td>
<td>Real provider, Amazon EC2 + HttpPerf + Media Wiki</td>
</tr>
<tr>
<td>[52]</td>
<td>RightScale + MA to performance metric</td>
<td>H</td>
<td>R</td>
<td>Number of active sessions</td>
<td>Custom tool</td>
<td>—</td>
<td>Synthetic, Different number of HTTP clients</td>
<td>Custom testbed, Xen + custom collaborative web application</td>
</tr>
<tr>
<td>[73]</td>
<td>RightScale + TS: LR and AR(1)</td>
<td>H</td>
<td>R/P</td>
<td>Request rate, CPU load</td>
<td>Simulated.</td>
<td>—</td>
<td>Synthetic, Three traffic patterns: weekly oscillation, large spike and random</td>
<td>Custom simulator, tuned after some real experiments</td>
</tr>
<tr>
<td>[59]</td>
<td>RightScale</td>
<td>H</td>
<td>R</td>
<td>CPU load</td>
<td>Amazon CloudWatch</td>
<td>—</td>
<td>Real, World Cup 98</td>
<td>Real provider, Amazon EC2 + RightScale (PaaS) + a simple web application</td>
</tr>
<tr>
<td>[96]</td>
<td>RightScale + Strategy-tree</td>
<td>H</td>
<td>R</td>
<td>Number of sessions, CPU idle</td>
<td>Custom tool, 4 minutes.</td>
<td>—</td>
<td>Real, World Cup 98</td>
<td>Real provider, Amazon EC2 + RightScale (PaaS) + a simple web application</td>
</tr>
<tr>
<td>[81]</td>
<td>Rules</td>
<td>V</td>
<td>R</td>
<td>CPU load, memory bandwidth, storage</td>
<td>Simulated.</td>
<td>—</td>
<td>Synthetic</td>
<td>Custom simulator, plus Java rule engine Drools</td>
</tr>
<tr>
<td>[77]</td>
<td>Rules</td>
<td>V</td>
<td>R</td>
<td>CPU load</td>
<td>Simulated, 1 minute</td>
<td>Response time</td>
<td>Real, ClarkNet</td>
<td>Custom simulator</td>
</tr>
</tbody>
</table>

Table rows are as follow: (1) The reference to the reviewed paper. (2) A short description of the proposed technique. (3) The type of auto-scaling: horizontal (H) or vertical (V). (4) The reactive (R) and/or proactive (P) nature of the proposal. (5) The performance metric or metrics driving auto-scaling. (6) The monitoring tool used to gather the metrics. The remaining three fields are related to the environment in which the technique is tested. (7) The metric used to verify SLA compliance. (8) The workload applied to the application managed by the auto-scaler. (9) The platform on which the technique is tested.

High level elasticity control in SYBL
(http://tuwiendsg.github.io/iCOMOT/)

#SYBL.CloudServiceLevel
Cons1: CONSTRAINT responseTime < 5 ms
Cons2: CONSTRAINT responseTime < 10 ms
WHEN nbOfUsers > 10000
Str1: STRATEGY CASE fulfilled(Cons1) OR
fulfilled(Cons2): minimize(cost)

#SYBL.ServiceUnitLevel
Str2: STRATEGY CASE ioCost < 3 Euro:
maximize(dataFreshness)

#SYBL.CodeRegionLevel
Cons4: CONSTRAINT dataAccuracy>90%
AND cost<4 Euro


A quick check: if you want to allow the developer to specify elasticity in his/her source code, e.g., Java, what would be your solution?
Play elasticity from the ground?

- Focus on assignment 3
- Use this trivial code:
  https://github.com/linhsolar/distributedsystemsexamples/tree/master/simple-upload-elasticity

  to write a simple yet full feature of elasticity uploading example
VIRTUALIZATION AND ELASTICITY FOR IMPLEMENTING PERFORMANCE PATTERNS
Design for handling failures

- Resource failures
  - Problems with CPUs, networks, machines, etc.
  - → other dependent services failures
- Scopes: with an enterprise, within a data center, across multiple sites, across multiple infrastructures provided by different providers, etc.
- Our design must be ready to handle such failures
- Using virtualization and elasticity techniques to deal with issues
- Relying on best practices
Examples of best practices when using Amazon services

- Using Elastic IPs
- Utilize resources from multiple zones
- Maintain Amazon virtual machines
- Use Amazon Cloudwatch for monitoring
- Automatically make snapshots of VMs
- Automatically backups

Source: https://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf
Recall this case

Change the way to handle client requests outside the service and within the service
Which are possible solutions?

- Throttling
- Queue-based load leveling within the service
- Multiple instances and queues
- Multiple instances and elastic resources
- Circuit breaker to deal with failures
- You name it
Throttling

Disable too many access and disable unessential services

Code: http://www.django-rest-framework.org/api-guide/throttling/#how-throttling-is-determined
Example

Custom domain indicates the tenant for this user

http://surveys.adatum.com
5 requests per second

http://surveys.fabrikam.com
10 requests per second

http://surveys.contoso.com
150 requests per second

Error: "Throttled"

Using tasks and queue-based load leveling pattern

Examples of queue-based load leveling pattern

Using multiple instances of services and queues

How do we control these instances in an efficient way?

Load balancing and elastic resources -- recall

Figure source: http://queue.acm.org/detail.cfm?id=1971597
Load balancing and elastic resources -- Concepts

- Using loadbalancer for a group of resources

- Load balancer can monitor instances and send request to healthy instances but what if we still need more instances?

- Auto-scaling
Examples

Amazon services

Google (from console.cloud.google.com)

They are programming tasks

Sources: http://docs.aws.amazon.com/autoscaling/latest/userguide/policy_creating.html
Examples from Amazon services

**Increase Group Size**
- **Name:** AddCapacity
- **Execute policy when:** AddCapacityAlarm Edit Remove
  - Breaches the alarm threshold: CPUUtilization >= 80 for 300 seconds for the metric dimensions AutoScalingGroupName = my-asg
- **Take the action:** Add 30 percent of group when 80 <= CPUUtilization < +Infinity
- **Add step:** Add instances in increments of at least 7 instance(s)
- **Instances need:** 300 seconds to warm up after each step

**Decrease Group Size**
- **Name:** DecreaseCapacity
- **Execute policy when:** DecreaseCapacityAlarm Edit Remove
  - Breaches the alarm threshold: CPUUtilization <= 40 for 300 seconds for the metric dimensions AutoScalingGroupName = my-asg
- **Take the action:** Remove 2 instances when 40 >= CPUUtilization > -Infinity
- **Add step:**

Sources: [http://docs.aws.amazon.com/autoscaling/latest/userguide/policy_creating.html](http://docs.aws.amazon.com/autoscaling/latest/userguide/policy_creating.html)
What if service operations fail due to unexpected problems or cascade failures (e.g. busy $\rightarrow$ timeout)
- Let the client retry and serve their requests may not be good

$\rightarrow$ Circuit breaker pattern prevents clients to retry an operation that would likely fail anyway and to detect when the operation failure is resolved.
Circuit breaker pattern

http://martinfowler.com/bliki/CircuitBreaker.html

Open Case Study for recap

- Multiple topics
- Amount of data per topic varies
- Should not have duplicate data in database

- Should I use docker? VMs?
- Where elasticity can be applied?
- Topic/data distribution to ingest clients?
Summary

- Modern distributed applications should consider underlying computing resources
  - Incorporate features to leverage virtualization and elasticity at runtime through programming tasks
- Elasticity and virtualization enable robust, efficient and reliable distributed applications
- They can also simplify the development and operation activities.
- Do exercises by examining examples in this lecture → e.g., providing your dockers for next year students
Further materials

- https://www.computer.org/web/the-clear-cloud/content?g=7477973&type=blogpost&urlTitle=performance-patterns-in-microservices-based-integrations
Thanks for your attention

Hong-Linh Truong
Faculty of Informatics, TU Wien
hong-linh.truong@tuwien.ac.at
http://www.infosys.tuwien.ac.at/staff/truong
@linhsolar