Cloudmesh Microservices for Making Analytics as Services

Gregor von Laszewski
laszewski@gmail.com

Disclaimer, sound has not been updated on all slides yet
BigData Analysis on Multicloud: Motivations

Capacity Federation

Technology Integration

Multicloud

- SaaS
- PaaS
- IaaS
- BareMetalaaS
- ...aaS
- HPCaaS

Future Systems

Chameleon Cloud

Google

Oracle

AWS

Azure

Chameleon Cloud

Multicloud

HPC
Big Data Analysis on Multicloud: Motivations

Service Robustness

Technology Federation

Compute Services

Containers
Virtual Machines
HPC

Multicloud

Azure
AWS

Open Stack
Oracle
Google

AWS
Azure

lazewski@gmail.com,
https://cloudmesh.github.io/cloudmesh-manual/
Some Requirements

**Motivation**
- Price Transparency
- Availability
  - Fault Tolerance
- Capacity
  - Resource Limitations
- Features within the cloud
  - Hybrid Clouds
- Independence:
  - Avoid vendor lock in

**Requirements**
- Accessible
- Ease of use
- Integrated
- Flexible
- Support multiple user community types
  - Enduser
  - Administrator
  - Developers

laszewski@gmail.com,
https://cloudmesh.github.io/cloudmesh-manual/
Goals: Cloudmesh

• Goal develop a mashup of cloud services to build a service mesh for analysis

• Functionality
  • IaaS Mesh
    • Compute – VMs
      • **Aws, Azure, OpenStack, Google, Oracle, ...**
      • Containers (Docker, Kubernetes)
      • local / SSH
      • SLURM
      • Create a virtual cluster from them
    • Data - **Aws, Azure, OpenStack, Google, Box, (github), (iCloud),**
      • Create a virtual directory that can store files as mashup everywhere
  • Service Generator
    • Take Python function and deliver OpenAPI spec and service automatically

• Backend
  • Deliver functionality them through a Python API
  • Deliver functionality them through Commandline and a Shell
  • Deliver REST Services through OpenAPI
    • This allows other language interfaces to be delivered
Cloudmesh Layered Architecture View

• Makes use of advanced cyber infrastructure and platforms
• Has deployment features
• Exposes functionality through
  • API
  • CLI
  • REST
  • (Client Portal)
  • Integration of AI as a composable items (functions and messaging between them)
• Simple use and deploy from the commandline
  • cms cloud=AWS
  • cms vm start
• Easy expandable
  • cms sys command generate NAME
    • generates a command with name

laszewski@gmail.com,
https://cloudmesh.github.io/cloudmesh-manual/
Cloudmesh Virtual Directory

- GUI to easily visualize
- CLI command line tool to easily script
- REST service
  - Provide ability to expose the functionality in a portable fashion
- Python APIs
  - Provide functionality for the implementation
  - Used to implement the REST Service
  - Backend Database is MongoDB
- Scheduler
  - Integration of a scheduler to automate file placement on Cloud Services
  - A Policy defines the scheduling of files to Resources

Inputs

File Put REST Service Scheduler Policy

file:
  name: file1
  endpoint:
    - file1
    - filen

file:
  name: filen
  endpoint:

directory:
  name:
  parent:
  files:
    - file1
    - filen

Outputs

Aws Azure Google Drive XSEDE Box iCloud VirtualBox

laszewski@gmail.com, https://cloudmesh.github.io/cloudmesh-manual/
Cloudmesh Virtual Cluster

- GUI to easily visualize
- CLI command line tool to easily script
- REST service
  - Provide ability to expose the functionality in a portable fashion
- Python APIs
  - Provide functionality for the implementation
  - Used to implement the REST Service
  - Backend Database is MongoDB
- Scheduler
  - Integration of a scheduler to automate compute placement on Cloud Services
  - A Policy defines the scheduling if files to Resources

Inputs

Execu

tion

Put

REST

Service

Scheduler

Policy

laszewski@gmail.com,
https://cloudmesh.github.io/cloudmesh-manual/
Cloudmesh Shell – Super Simple to Boot VM

$ cms set cloud=NAME
$ cms vm boot
$ cms vm ssh
$ cms vm delete

• cloudmesh.yaml
• Prepare defaults
• Boot
• Login
• Management

…
Cloudmesh Shell – Simple to Manage Hybrid Clouds

$ cms set cloud=aws
$ cms vm boot

$ cms set cloud=chameleon
$ cms vm boot

$ cms set cloud=azure
$ cms vm boot

• Boot Cloud A
• Boot Cloud B
• Boot Cloud C
Jupyter Integration

Cloudmesh can easily be integrated into jupyter

The command shell is readily accessible via an API call

```python
In [1]: from cloudmesh.cloud import Shell

# ----------------------------------------
# Cloudmesh
# ----------------------------------------

In [2]: r = Shell.run("ls")

In [3]: print(r)

cloudmesh.ipynb

In [4]: r = Shell.run("help")

In [5]: print(r)

Documented commands (type help <topic>):
----------------------------------------
EOF    container help    key    q    shell stopwatch vm
admin  data       image  man    quit   sleep   sys   wps
banner default info    open    register source    var
break  echo       init    pause    set    ssh      var
commands flavor       inventory plugin   service start    stop
config  group    ip      provider set
----------------------------------------

Timer: 0.0089s (help)

In [7]: r = Shell.run("set cloud=chameleon")

In [8]: r = Shell.run("set refresh=True")

In [9]: r = Shell.run("flavor list")

In [10]: print(r)

cloud chameleon
----------------------------------------
<table>
<thead>
<tr>
<th>Name</th>
<th>VCPUs</th>
<th>RAM</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl.tiny</td>
<td>1</td>
<td>512</td>
<td>1</td>
</tr>
<tr>
<td>nl.small</td>
<td>1</td>
<td>2048</td>
<td>20</td>
</tr>
<tr>
<td>nl.medium</td>
<td>2</td>
<td>4096</td>
<td>40</td>
</tr>
<tr>
<td>nl.large</td>
<td>4</td>
<td>8192</td>
<td>80</td>
</tr>
<tr>
<td>nl.xlarge</td>
<td>8</td>
<td>16384</td>
<td>160</td>
</tr>
<tr>
<td>storage.medium</td>
<td>1</td>
<td>4096</td>
<td>2048</td>
</tr>
<tr>
<td>nl.xxlarge</td>
<td>8</td>
<td>32768</td>
<td>160</td>
</tr>
<tr>
<td>nl.xxxlarge</td>
<td>16</td>
<td>32768</td>
<td>160</td>
</tr>
</tbody>
</table>
----------------------------------------

Timer: 1.9617s (flavor list)
```
Simple API

Super simple API that allows integration with jupyter notebooks very easily

In [1]: from cloudmesh.compute.vm.Provider import Provider

In [2]: provider = Provider(name="chameleon")

In [3]: flavors = provider.flavors()

In [4]: flavors[0]["name"]

Out[4]: 'ml.tiny'

In [5]: provider.Print(flavors)

+---------------------------------+----------+-------+-------+
<table>
<thead>
<tr>
<th>Name</th>
<th>VCPUS</th>
<th>RAM</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.tiny</td>
<td>1</td>
<td>512</td>
<td>1</td>
</tr>
<tr>
<td>ml.small</td>
<td>1</td>
<td>2048</td>
<td>20</td>
</tr>
<tr>
<td>ml.medium</td>
<td>2</td>
<td>4096</td>
<td>40</td>
</tr>
<tr>
<td>ml.large</td>
<td>4</td>
<td>8192</td>
<td>80</td>
</tr>
<tr>
<td>ml.xlarge</td>
<td>8</td>
<td>16384</td>
<td>160</td>
</tr>
<tr>
<td>storage.medium</td>
<td>1</td>
<td>4096</td>
<td>2048</td>
</tr>
<tr>
<td>ml.xxlarge</td>
<td>8</td>
<td>32768</td>
<td>160</td>
</tr>
<tr>
<td>ml.xxxlarge</td>
<td>16</td>
<td>32768</td>
<td>160</td>
</tr>
</tbody>
</table>
+---------------------------------+----------+-------+-------+
Cloudmesh Microservices

• [https://github.com/cloudmesh/cloudmesh-openapi](https://github.com/cloudmesh/cloudmesh-openapi)

• **Microservice** architecture:
  • structures an application as a collection of services that are
    • Maintainable
    • Testable
    • Loosely coupled
    • Independently deployable
    • Project a clear functionality

• Problem:
  • While working with professionals, researchers, and students, entry-level is still to high for many
Automated Microservice Generation

Python Program
- Function
- Class
- Python documentation

OpenAPI
- YAML

Microservice
- Python
- OpenAPI enhanced
- Documentation
- Security
- Data integration
Super simple to Generate a Microservice

• Manual page
  • cms help openapi

• Generate the Yaml file
  • cms openapi generate get_processor_name --filename=./tests/server-cpu/cpu.py
Simple Python Program as a Function

import ...

def get_processor_name() -> str:
    """
    The name of the processor
    :return: the name of the processor
    """

    command = "cat /proc/cpuinfo" all_info = subprocess.check_output(
        command, shell=True).strip().decode() for line in all_info.split("\n"):
        p = re.sub(".*model name.*:", ", ", line, 1)
    return jsonify(pinfo)
Generated YAML file

openapi: 3.0.0
info:
  title: get_processor_name
  description: "The name of the processor"
  version: "1.0"
servers:
  - url: http://localhost:8080/cloudmesh
    description: "The name of the processor"
paths:
  /get_processor_name:
    get:
      summary: "The name of the processor"
      description: "Optional extended description in CommonMark or HTML.
      operationId: cpu.get_processor_name

    responses:
      '200':
        description: "OK"
        content:
          text/plain:
            schema:
              type: string

Server code is automatically generated
Can be started with `cloudmesh openapi` command easily
Can be managed with the command
(start stop register)
Issue a Request

curl -X GET "http://localhost:8080/cloudmesh/get_processor_name"
   -H "accept: text/plain"

127.0.0.1 - - [07/Oct/2020 14:10:04] "GET /cloudmesh/get_processor_name HTTP/1.1" 200 -
{"model":"Intel(R) Core(TM) i7-7920HQ CPU @ 3.10GHz"}
Automated Documentation

Open http://localhost:8080/cloudmesh/ui
Summary

• Cloudmesh Multicloud Management
  • Managing multicloud compute resources
  • Managing multicloud data resources

• Cloudmesh OpenAPI
  • Create easily many services from just python functions no need to be a software architect,
  • Inexperienced users can do it in a day (minutes)
  • Cloudmesh Multicloud Management could be used to host the rest service in a cloud
  • Restservice can naturally be also hosted on singularity or kubernetes

laszewski@gmail.com,
https://cloudmesh.github.io/cloudmesh-manual/
How to find information: GitHub

• Cloudmesh is distributed in multiple repositories
  • https://github.com/cloudmesh

• Manual
  • https://github.com/cloudmesh/cloudmesh-manual
  • https://cloudmesh.github.io/cloudmesh-manual/

• Cloudmesh OpenAPI

• Nist: OpenAPI specifications
  • https://github.com/cloudmesh/cloudmesh-openapi