Cloudmesh Microservices for Making Analytics as Services

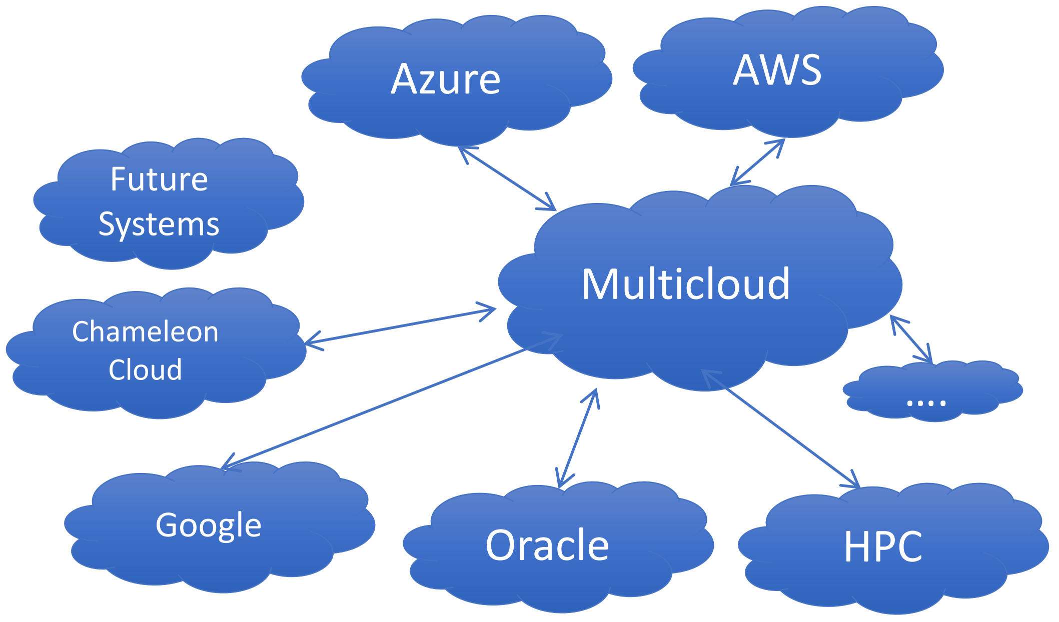
Gregor von Laszewski

laszewski@gmail.com

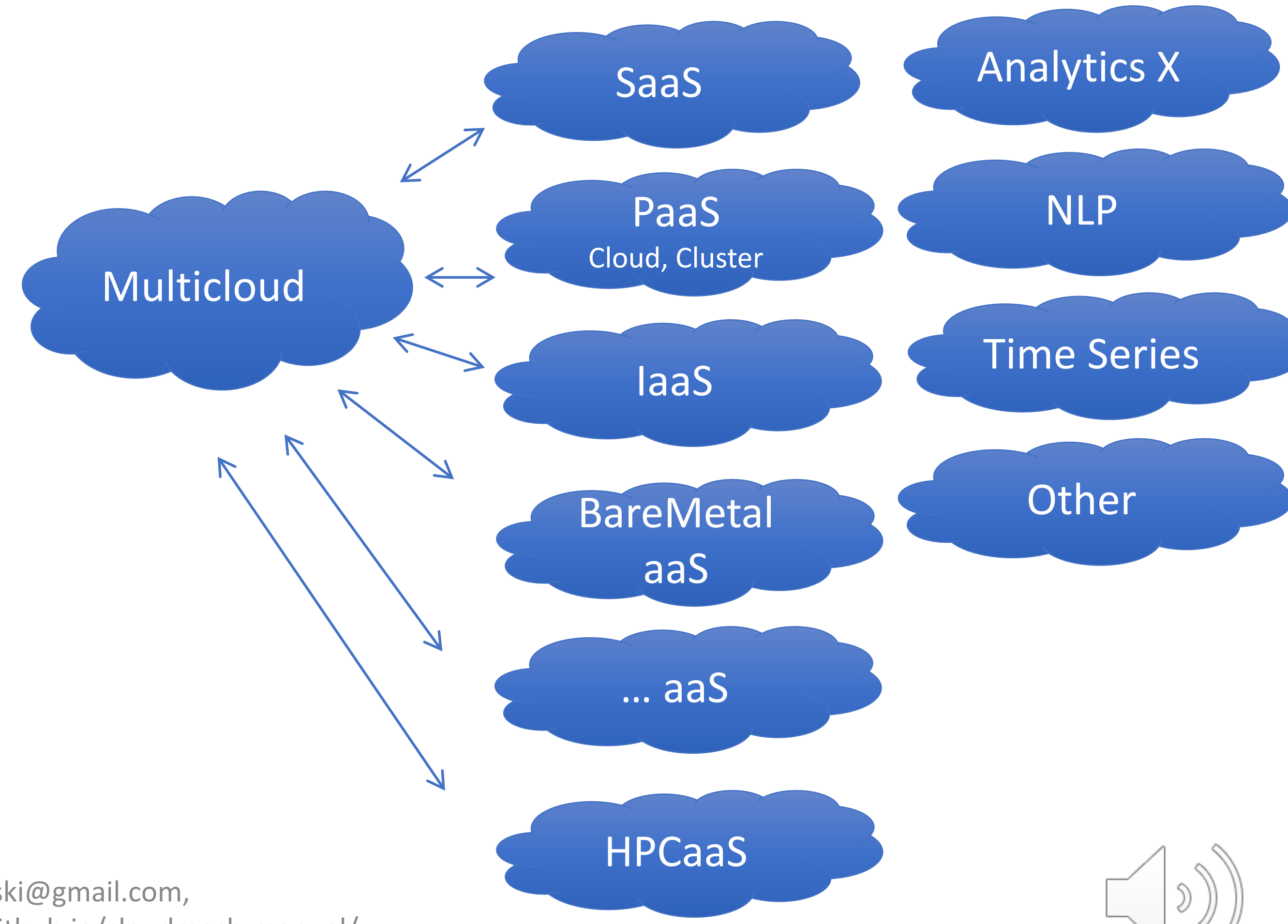
Disclaimer, sound has not been updated on all slides yet

BigData Analysis on Multicloud: Motivations

Capacity Federation



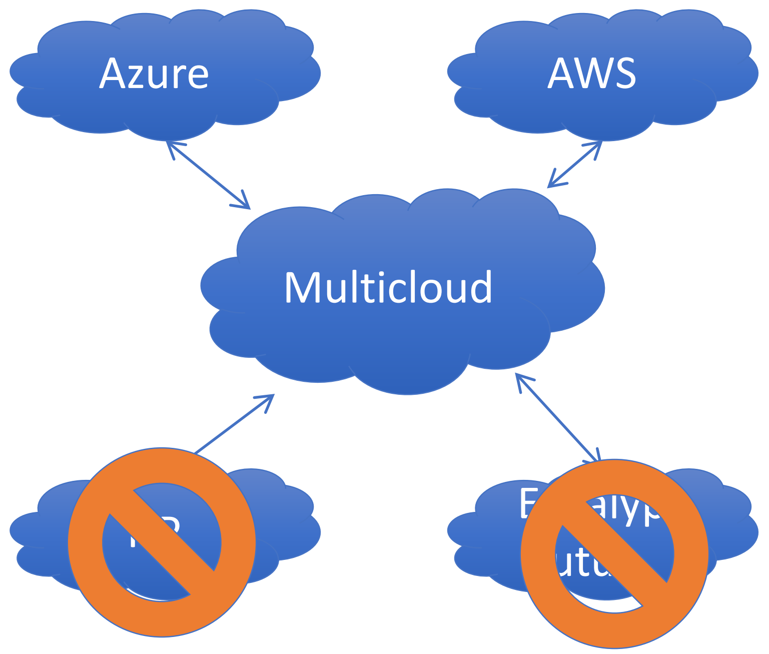
Technology Integration



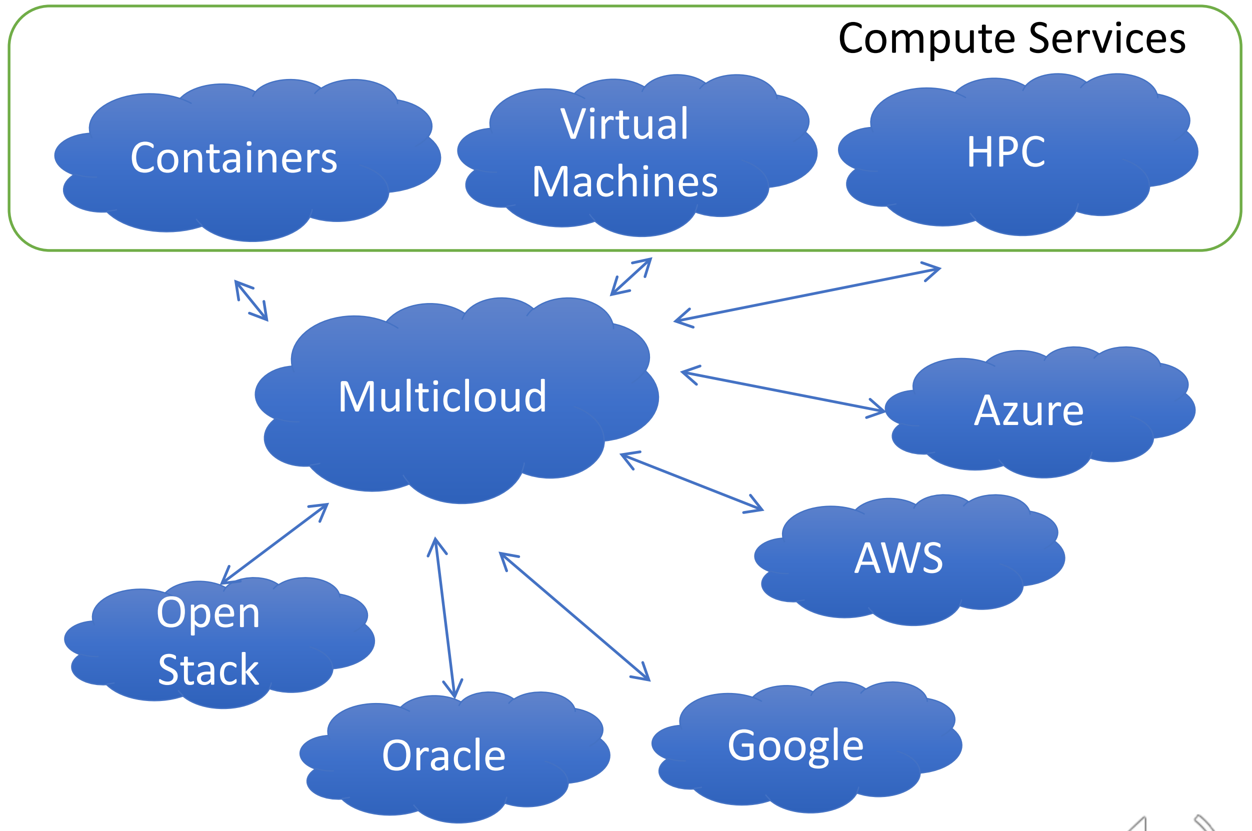
laszewski@gmail.com,
https://cloudmesh.github.io/cloudmesh-manual/

Big Data Analysis on Multicloud: Motivations

Service Robustness



Technology Federation



Some Requirements

Motivation

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

Requirements

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

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

→ Virtual Cluster

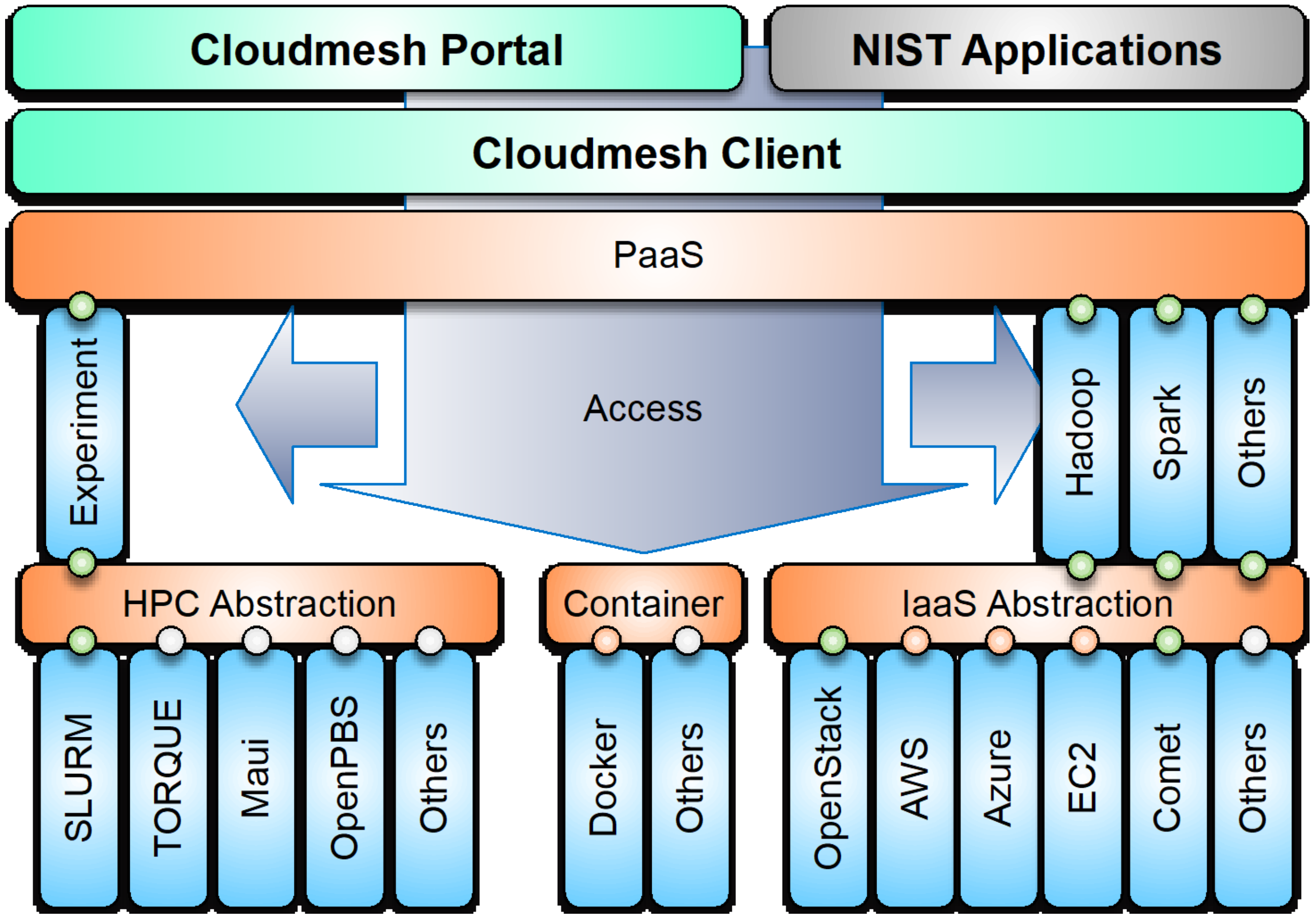
→ Virtual Directory

→ REST Service + OpenAPI

→ Easy Access

Cloudmesh Layered Architecture View

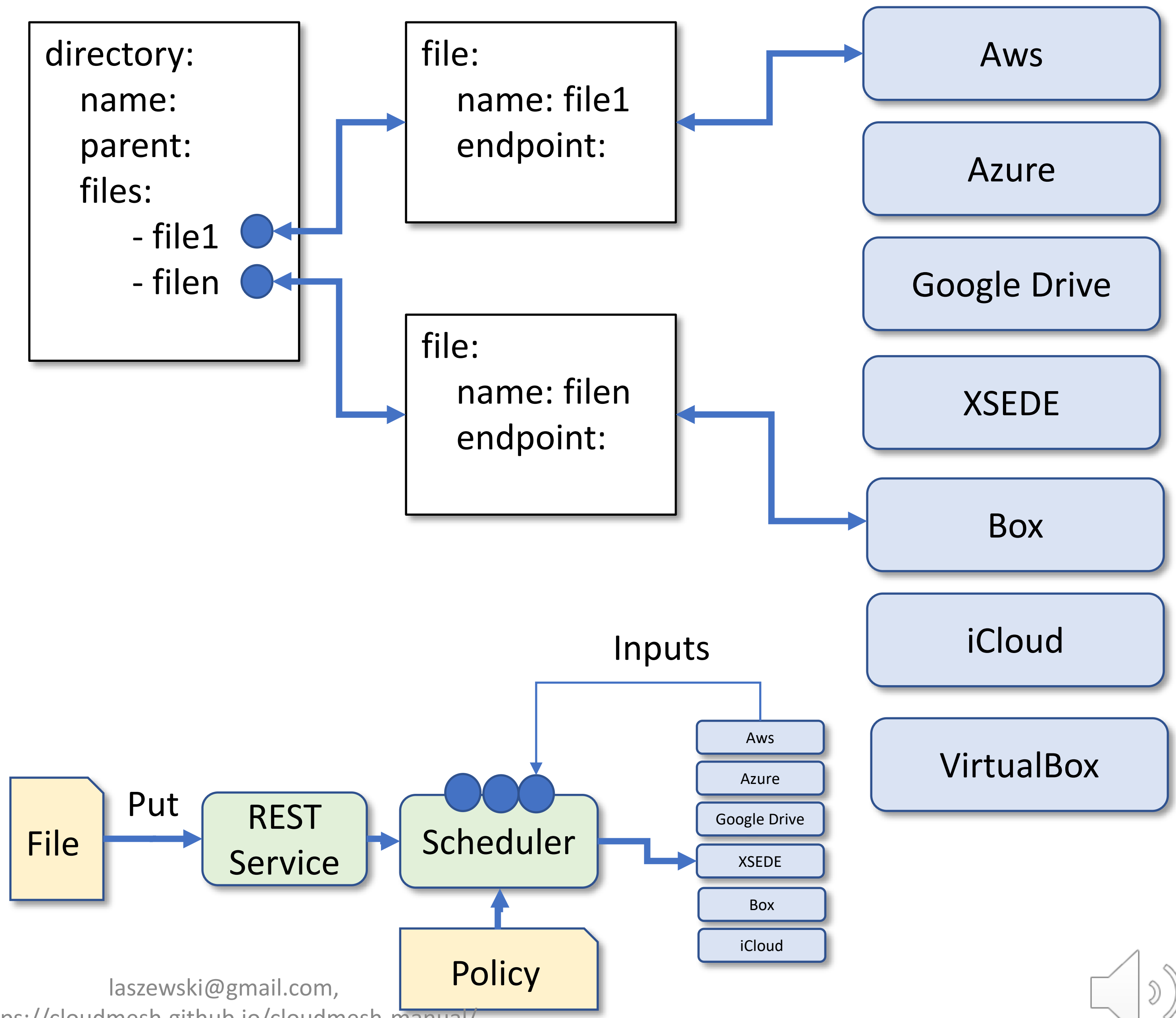
- Makes use of advanced cyber infrastructure and platforms
- Has deployment features
- Exposes functionality through
  - API
  - CLI
  - REST
  - (Client Portal)
  - Integration of AI as a composable items (functions and messaging between them)
- Simple use and deploy from the commandline
  - cms cloud=AWS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



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

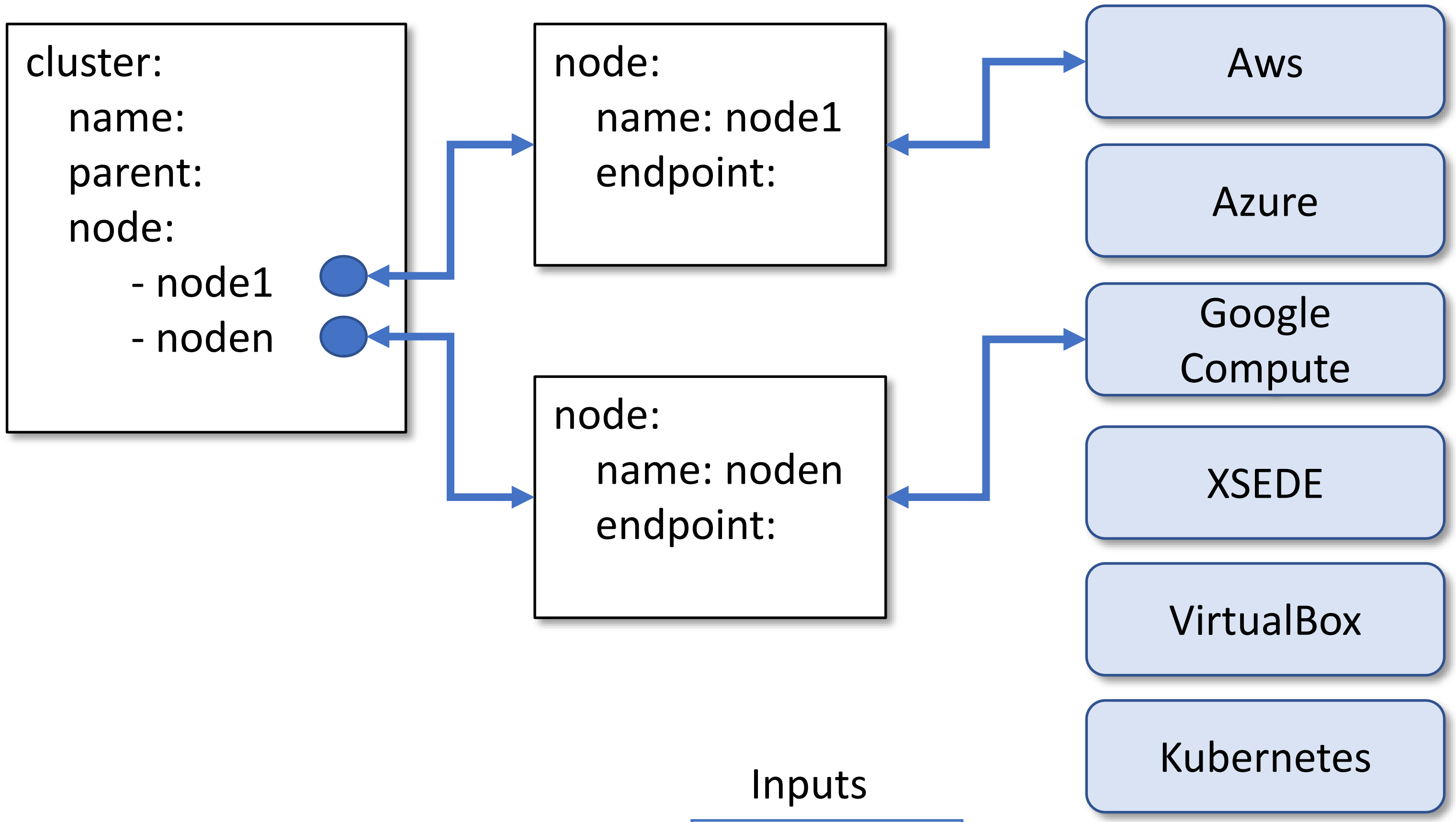
Cloudmesh Virtual Directory

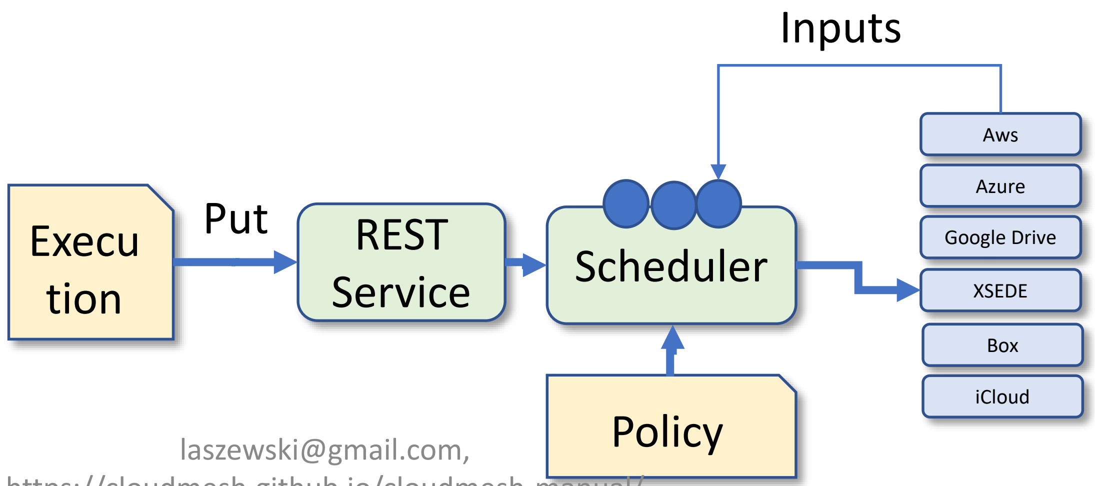
- GUI to easily visualize
- CLI command line tool to easily script
- REST service
  - Provide ability to expose the functionality in a protable fashion
- Python APIs
  - Provide functionality for the implementation
  - Used to implement the REST Service
  - Backend Database is MongoDB
- Scheduler
  -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



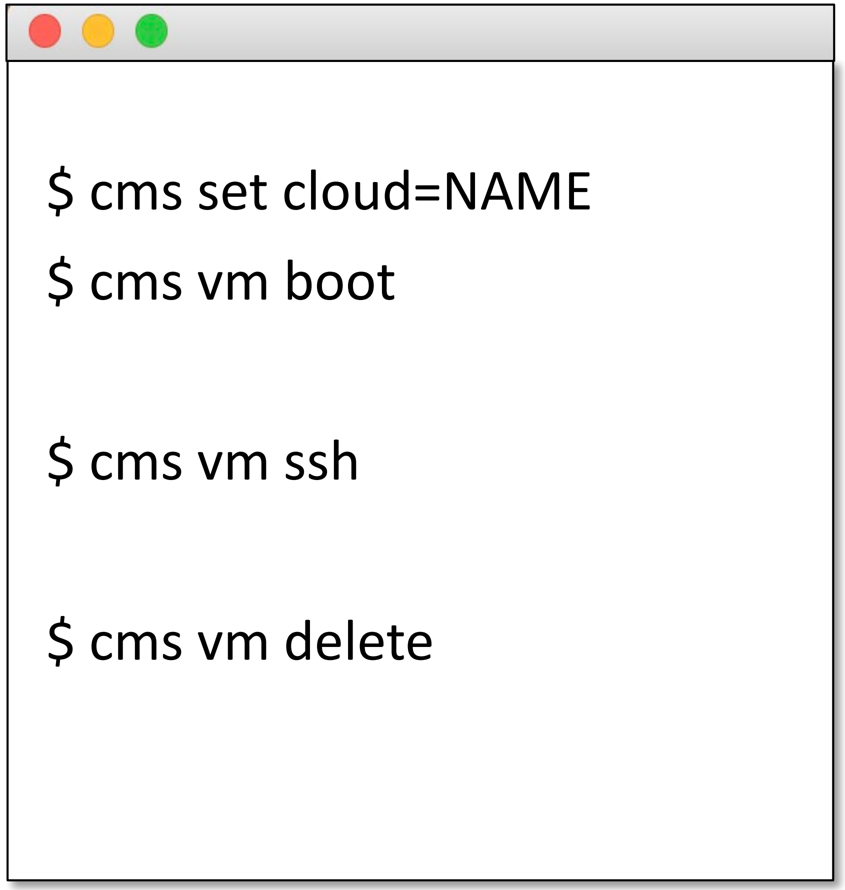
Cloudmesh Virtual Cluster

- 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

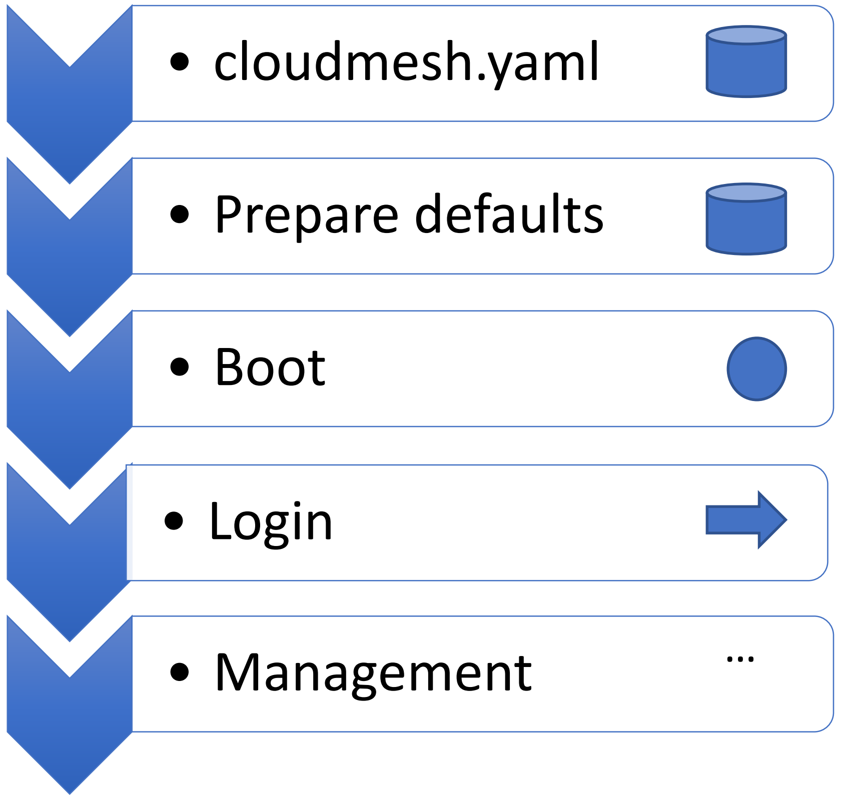




Cloudmesh Shell – Super Simple to Boot VM

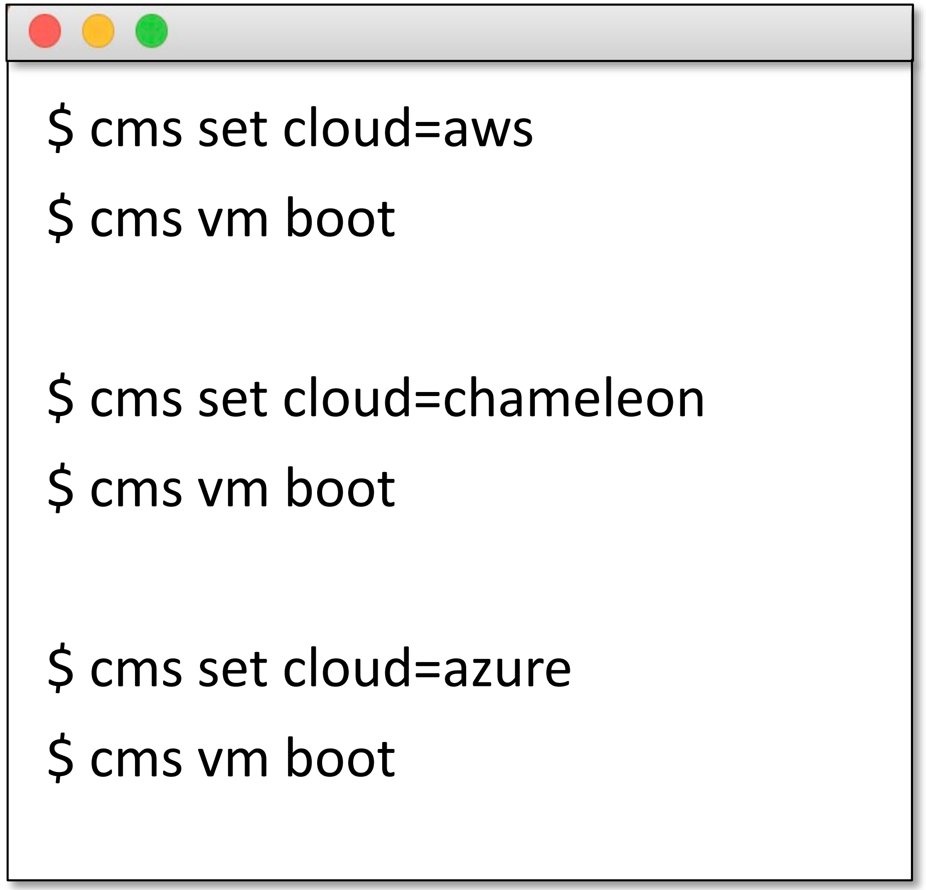


```
$ cms set cloud=NAME
$ cms vm boot

$ cms vm ssh

$ cms vm delete
```

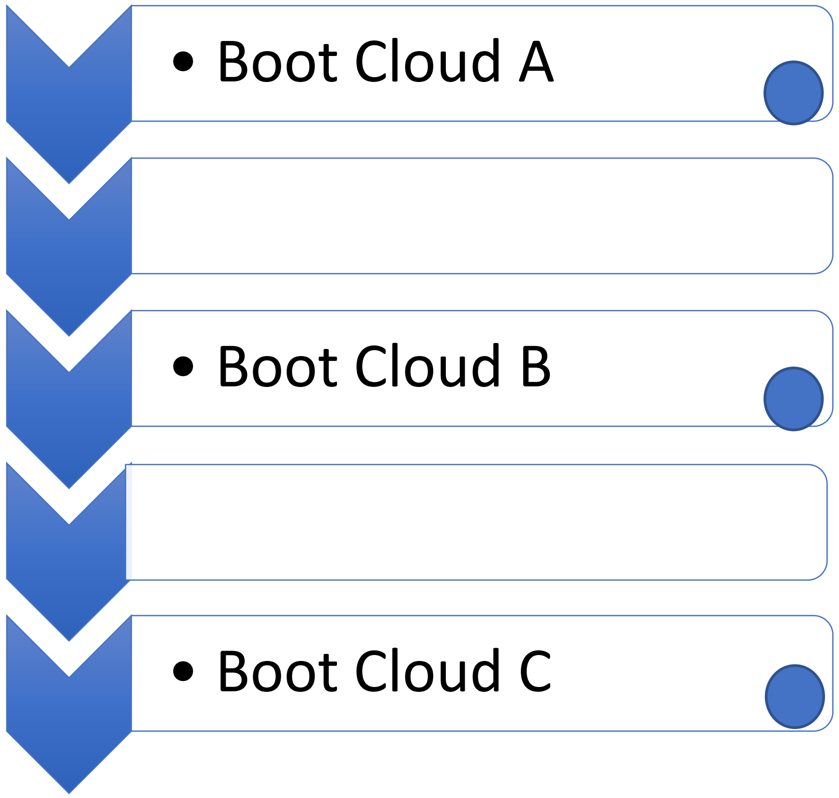


- cloudmesh.yaml
- Prepare defaults
- Boot
- Login
- Management …

Cloudmesh Shell – Simple to Manage Hybrid Clouds



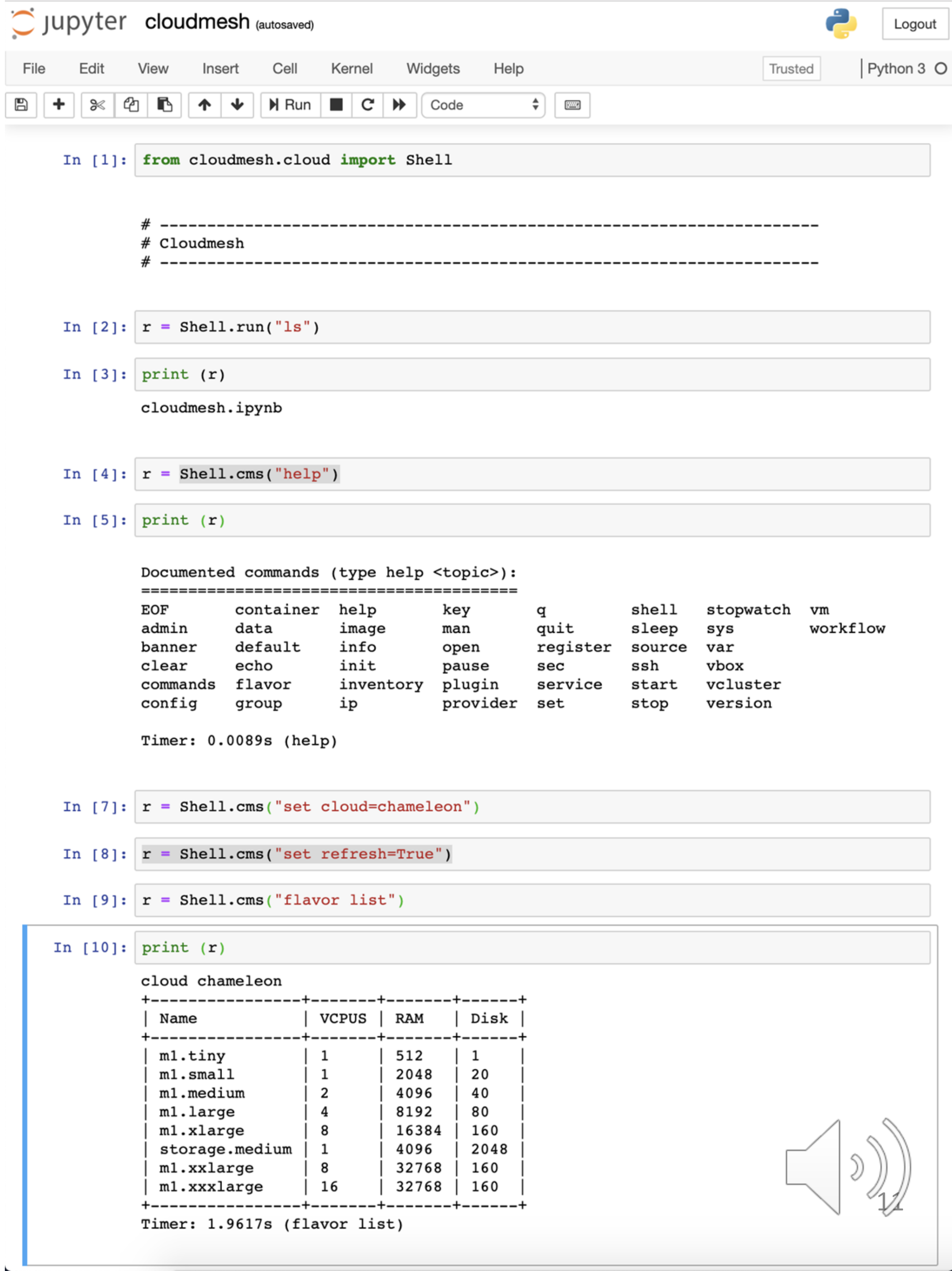
```
$ cms set cloud=aws
$ cms vm boot

$ cms set cloud=chameleon
$ cms vm boot

$ cms set cloud=azure
$ cms vm boot
```



Jupyter Integration

Cloudmesh can easily be integrated into jupyter

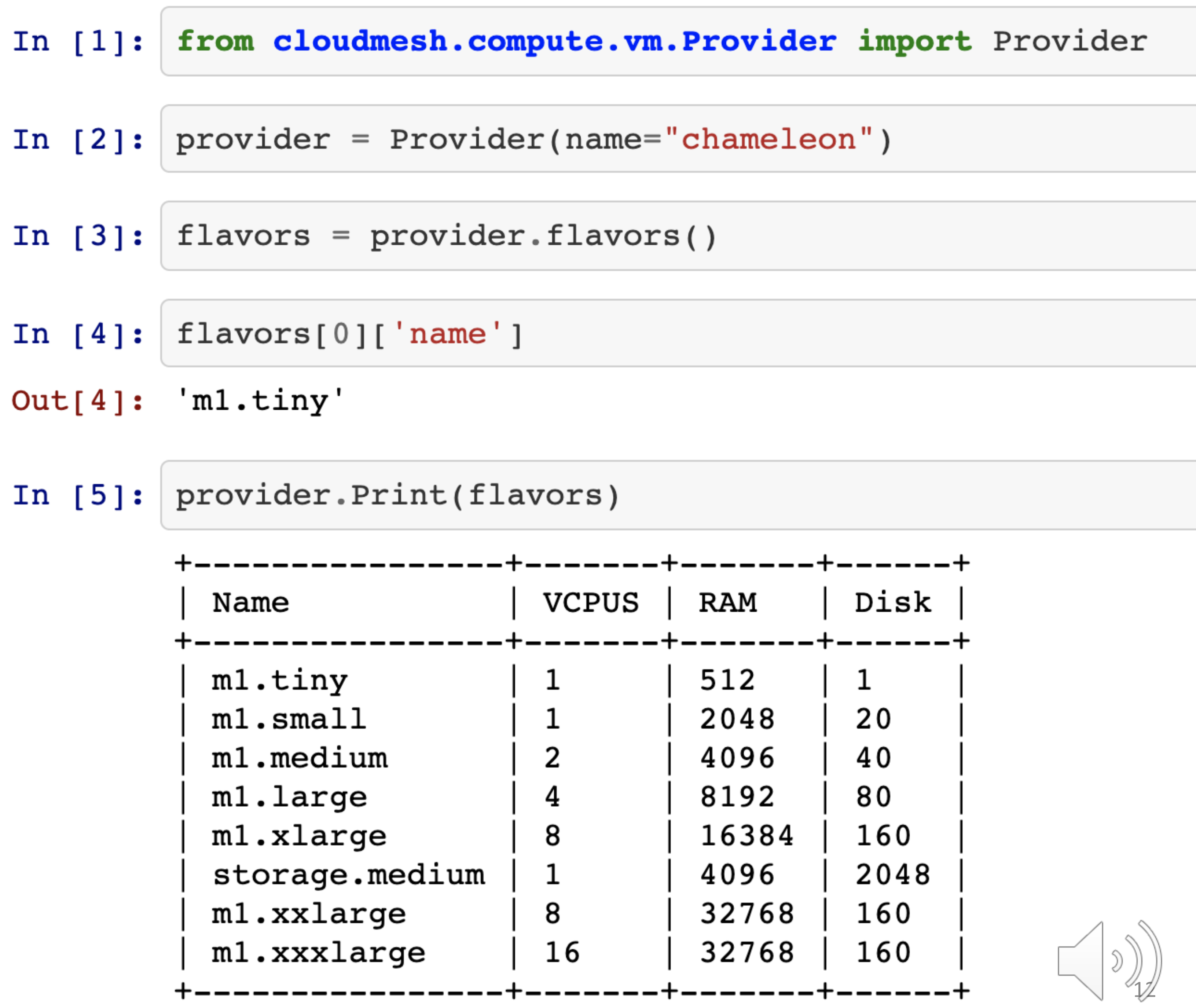
The command shell is readily accessible via an API call


jupyter cloudmesh (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

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

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

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

In [3]: print (r)

        cloudmesh.ipynb

In [4]: r = Shell.cms("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        workflow
        banner    default    info       open      register  source  var
        clear     echo       init       pause     sec       ssh     vbox
        commands  flavor     inventory  plugin    service   start   vcluster
        config    group      ip         provider  set       stop    version

        Timer: 0.0089s (help)

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

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

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

In [10]: print (r)

        cloud chameleon
        +----------------+-------+-------+------+
        | Name           | VCPUS | RAM   | Disk |
        +----------------+-------+-------+------+
        | m1.tiny        | 1     | 512   | 1    |
        | m1.small       | 1     | 2048  | 20   |
        | m1.medium      | 2     | 4096  | 40   |
        | m1.large       | 4     | 8192  | 80   |
        | m1.xlarge      | 8     | 16384 | 160  |
        | storage.medium | 1     | 4096  | 2048 |
        | m1.xxlarge     | 8     | 32768 | 160  |
        | m1.xxxlarge    | 16    | 32768 | 160  |
        +----------------+-------+-------+------+
        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]: 'm1.tiny'

In [5]: provider.Print(flavors)

+----------------+-------+-------+------+
| Name           | VCPUS | RAM   | Disk |
+----------------+-------+-------+------+
| m1.tiny        | 1     | 512   | 1    |
| m1.small       | 1     | 2048  | 20   |
| m1.medium      | 2     | 4096  | 40   |
| m1.large       | 4     | 8192  | 80   |
| m1.xlarge      | 8     | 16384 | 160  |
| storage.medium | 1     | 4096  | 2048 |
| m1.xxlarge     | 8     | 32768 | 160  |
| m1.xxxlarge    | 16    | 32768 | 160  |
+----------------+-------+-------+------+
```

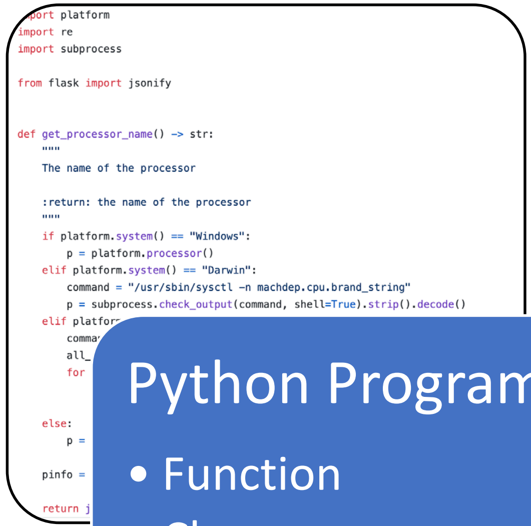
Cloudmesh Microservices

- 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 manys

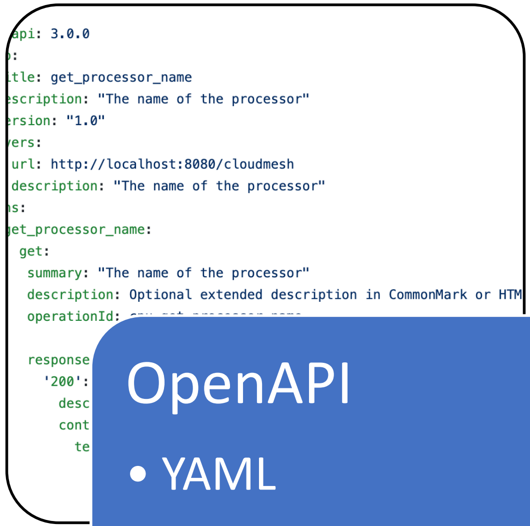
Automated Microservice Generation



```
port platform
import re
import subprocess

from flask import jsonify


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

    :return: the name of the processor
    """
    if platform.system() == "Windows":
        p = platform.processor()
    elif platform.system() == "Darwin":
        command = "/usr/sbin/sysctl -n machdep.cpu.brand_string"
        p = subprocess.check_output(command, shell=True).strip().decode()
    elif platfor
        comma
        all_
        for


    else:
        p =

    pinfo =

    return j
```

Python Program

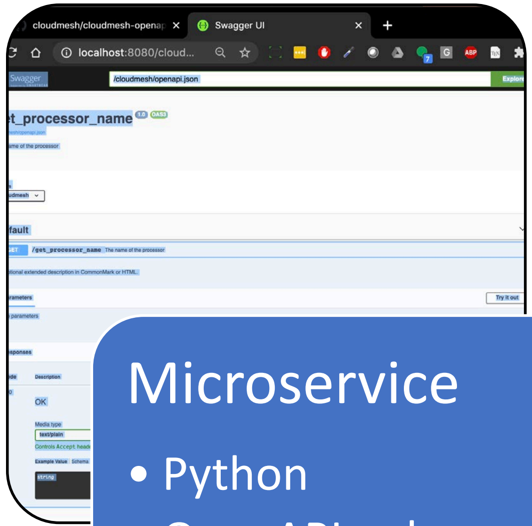
- Function
- Class
- Python documentation



```
api: 3.0.0
:
itle: get_processor_name
escription: "The name of the processor"
rsion: "1.0"
vers:
 url: http://localhost:8080/cloudmesh
 description: "The name of the processor"
ns:
et_processor_name:
  get:
   summary: "The name of the processor"
   description: Optional extended description in CommonMark or HTM
   operationId:

   response
     '200':
       desc
       cont
         te
```

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
```

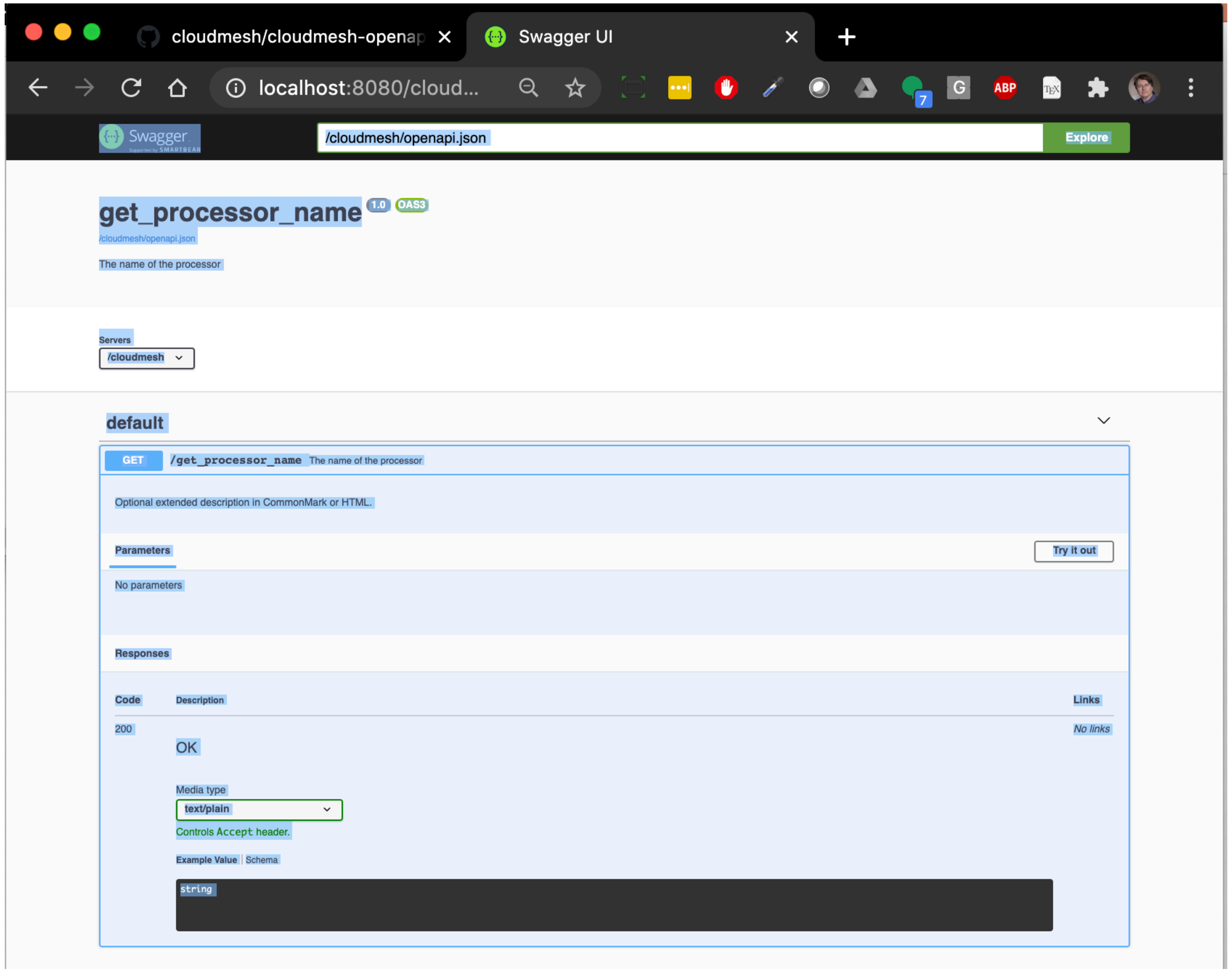
Server code is automatically generated
Can be started with cloudmesh openapi command easily
Can be managed with the command (start stop register)

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


cloudmesh/cloudmesh-openap
Swagger UI
localhost:8080/cloud...
/cloudmesh/openapi.json
Explore

get_processor_name 1.0 OAS3
/cloudmesh/openapi.json

The name of the processor

Servers
/cloudmesh

default

GET /get_processor_name The name of the processor

Optional extended description in CommonMark or HTML.

Parameters
Try it out

No parameters

Responses

Code	Description	Links
200	OK	*No links*

Media type
text/plain
Controls Accept header.

Example Value | Schema

```
string
```

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 puthon 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

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
  - https://github.com/laszewski/laszewski.github.io/raw/master/papers/vonLaszewski-openapi.pdf

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