Network services with CNSMO

Isart Canyameres, i2CAT

CYCLONE UCs Hackathon

Gif-sur-Yvette, FR, November 15th, 2016

OUTLINE

1.- Introduction to CNSMO component

2.- CNSMO architecture and technologies.

3.- CNSMO instantiation and deployment.

4.- Creating network services

5.- Deploying network services together with your application

6.- Foreseen next steps, features and services.

7.- QA

CNSMO introduction

1, 2, 3, 4, 5, 6, 7

CONCEPT

- The purpose of CNSMO *(zi-nís-mo)* is to offer network management capabilities to (multi)cloud applications

- Conceived as an application component, it automatizes the deployment of network services with short required input.

- Offered services (as to date):
  - Private network
  - Firewall
  - Load-balancer

Cyclone

CONCEPT (II)

- As an application developer, you can use CNSMO to easily enhance your application with network services to control:
  - VM network accessibility,
  - intra-site data access, and
  - inter-site data transfers

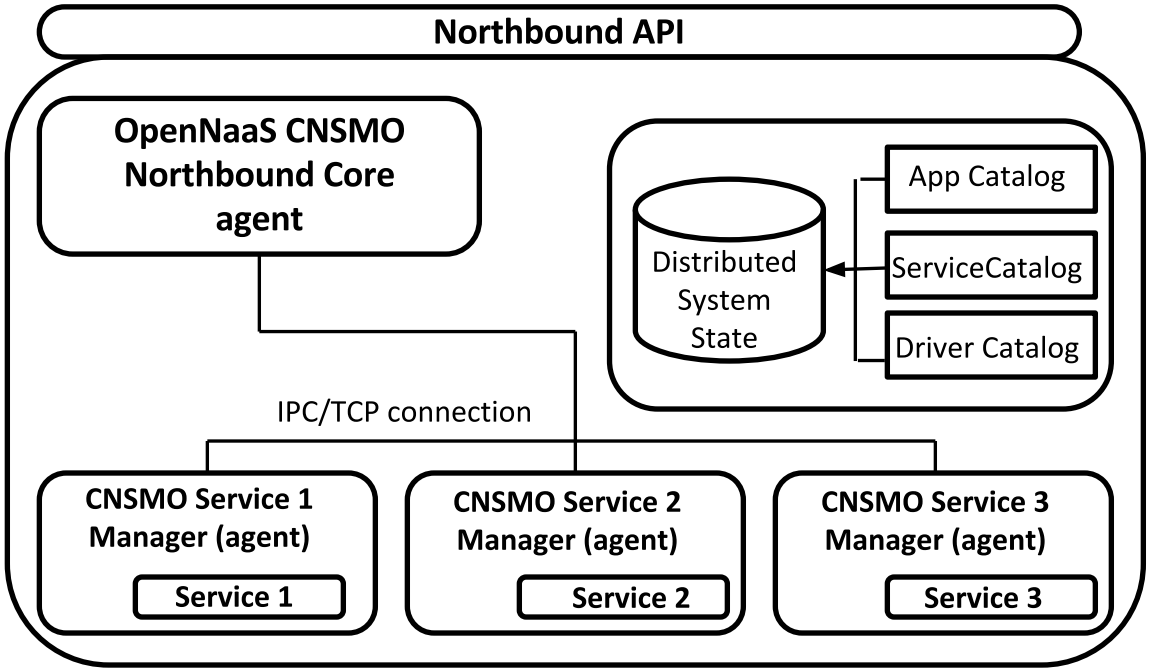
OPENNAAS CNSMO - OVERVIEW

- Lightweight micro-services framework.
- Responsible for deploying, configuring and running network services over distributed clouds.
- Use of network virtualization (VNF, VMs, containers) for services setup to increase scalability and automation.
- Minimize the impact on the user space.
- GPLv3 license.

Cyclone

http://opennaas.org/opennaas-cnsmo/

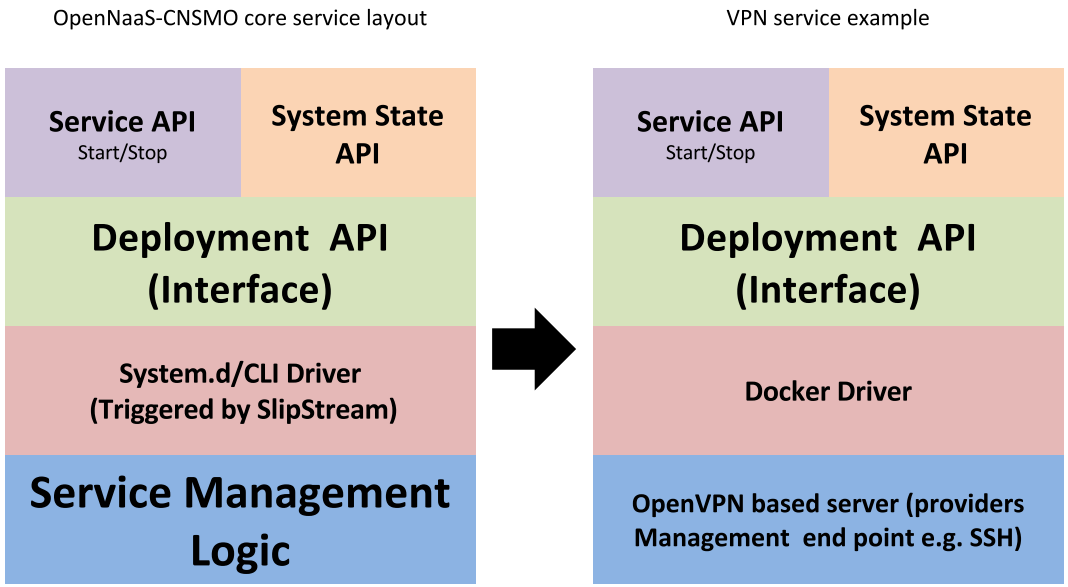
OpenNaaS CNSMO - ARCHITECTURE



- The CNSMO core
- CNSMO agents running networking services

NETWORK SERVICES (I)

- A "***Service***" in the context of CYCLONE is a set of modifications in a user deployment (provided by SlipStream) that provides an added value to the default functionalities of the different CSPs.



NETWORK SERVICES (II)

- CNSMO defines a service through a service metadata, including:
  - serviceId
  - trigger
  - resources (files)
  - dependencies
  - endpoints (exposed URIs)
  - deployer (driver)

OPENNAAS CNSMO (III) – ADDED VALUE

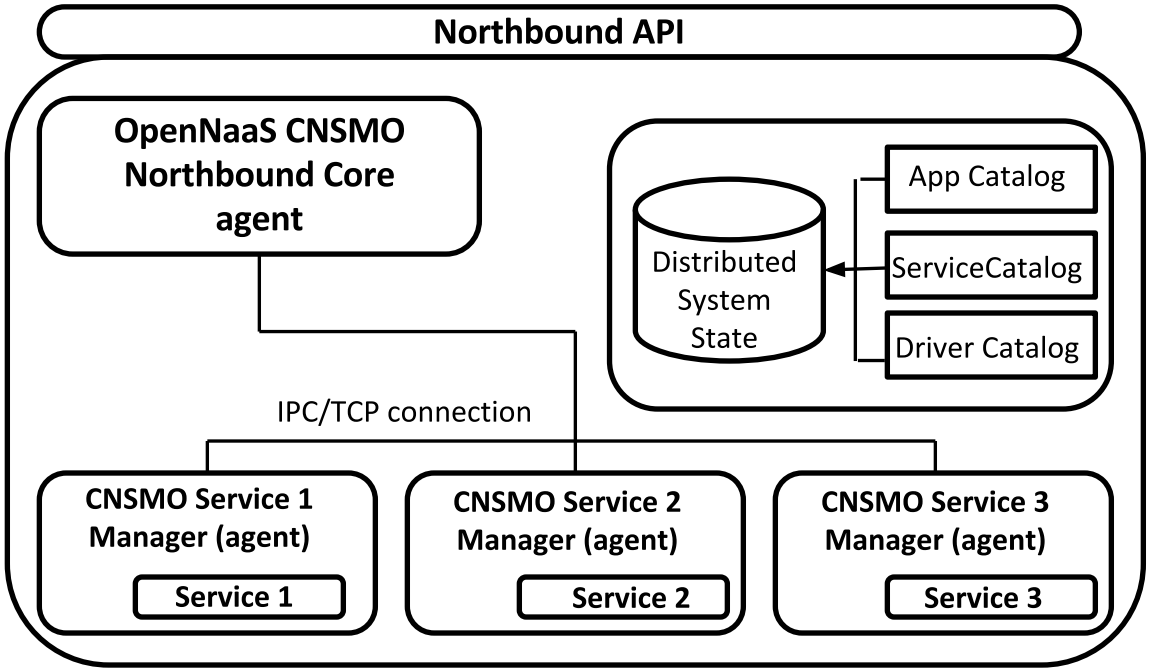
- Couple cloud applications together with network services.
- Automation of processes.
- Ability to potentially run on any Cloud.
- Lightweight framework.
- Barely intrusive with the applications and running on the user space.
- Provisioning of Integrated network services (selection enabled).
- Extends SlipStream offer (as part of its app store).
- Typical MUST-BE features:
  - Scalability
  - Resiliency

Cyclone

CNSMO architecture and tech

1, **2**, 3, 4, 5, 6, 7

CNSMO ARCHITECTURE



- The CNSMO core
- CNSMO agents running networking services

CNSMO TECHNOLOGIES

- CNSMO is a custom **python** module
- Uses **redis** as a Distributed State Storage
- Services are deployed via **git**

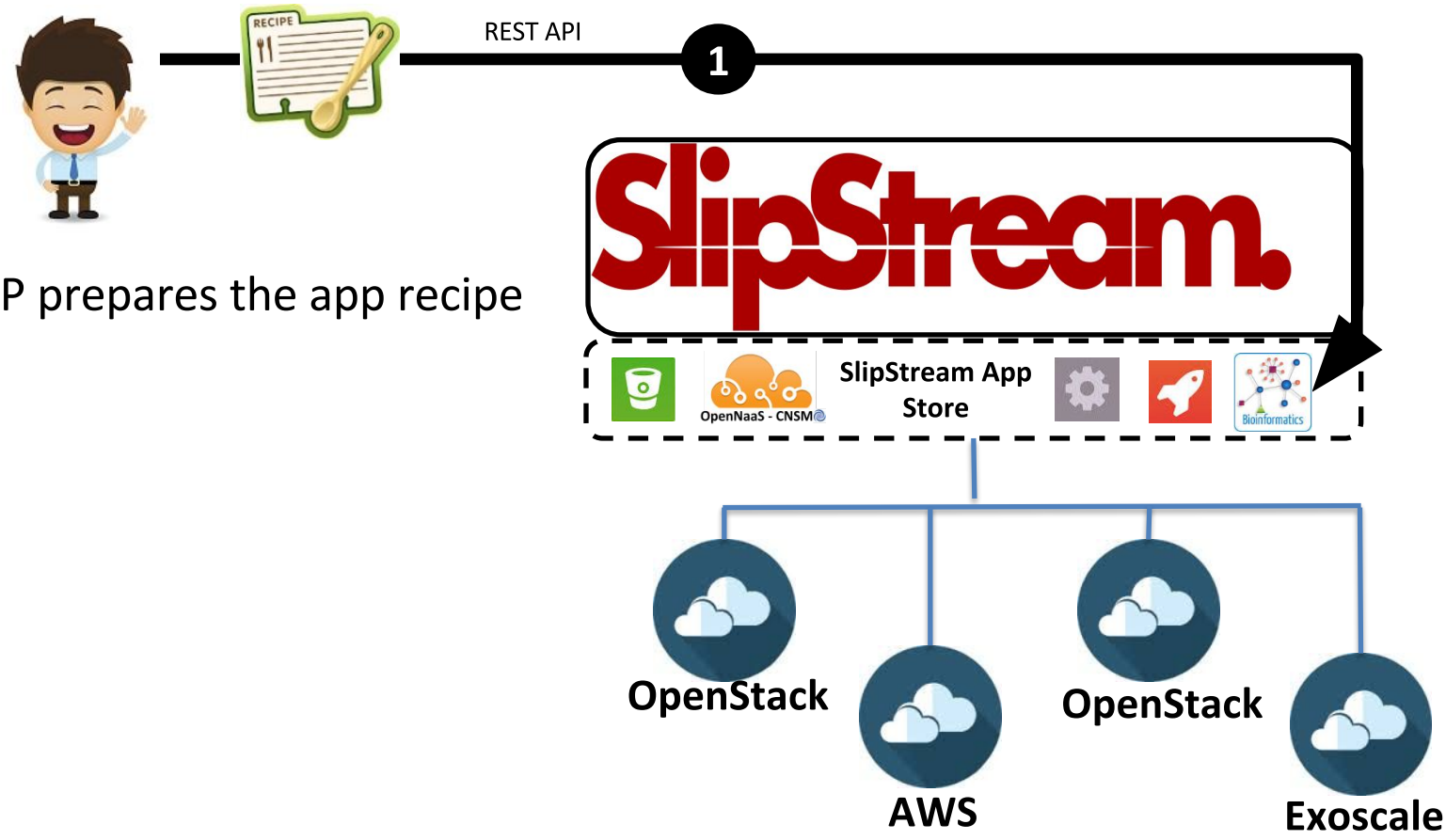
- Services are triggered via **bash**
- Services run in **docker** containers
- Services are managed through **REST APIs** provided via **flask**
- Services commonly use **jinja** as a configuration file generator
- Configuration files are transmitted over **HTTP**

- Alternative custom service implementations are supported

Running CNSMO

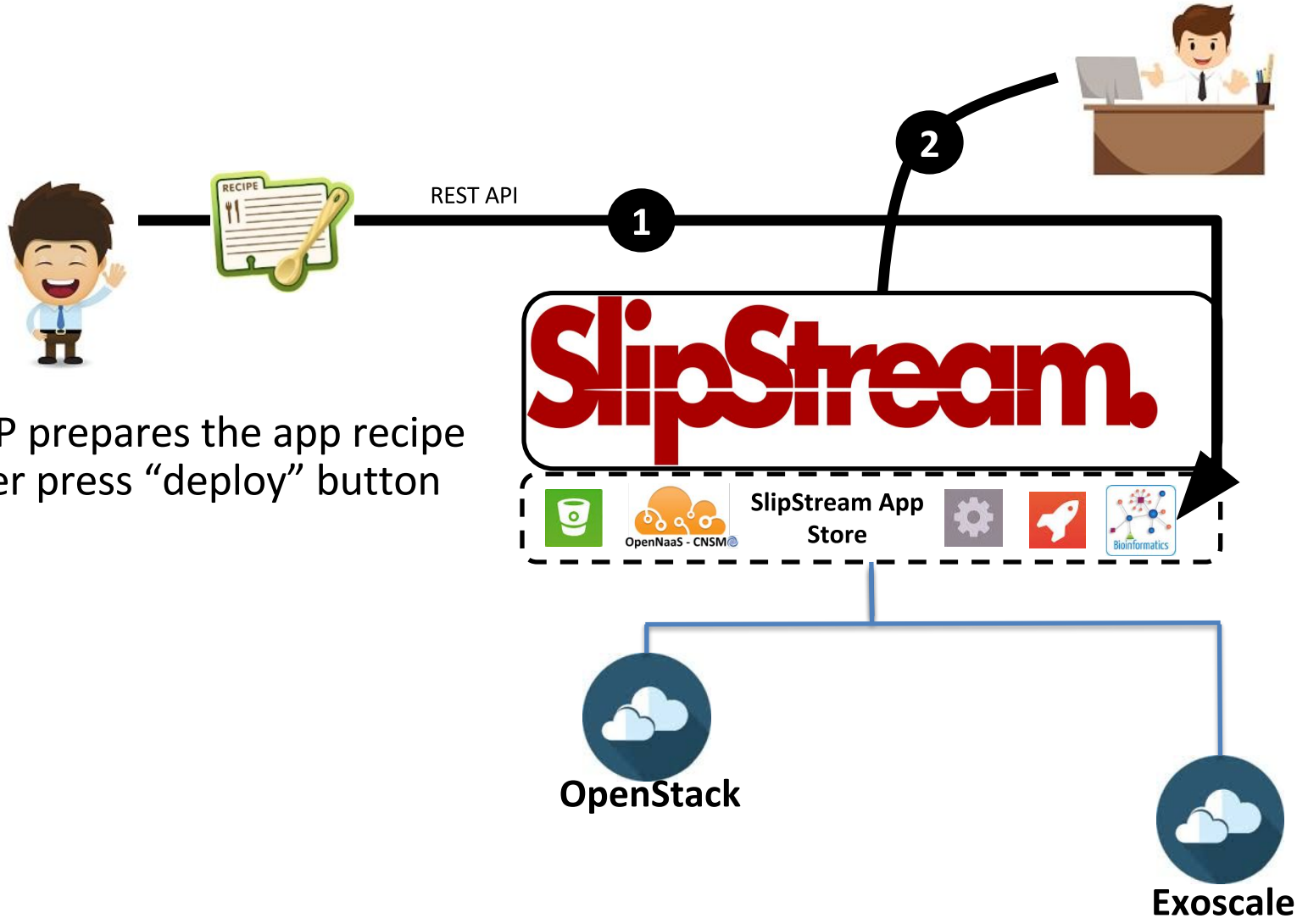
1, 2, **3**, 4, 5, 6, 7

INTEGRATION OF NETWORK SERVICES (I)



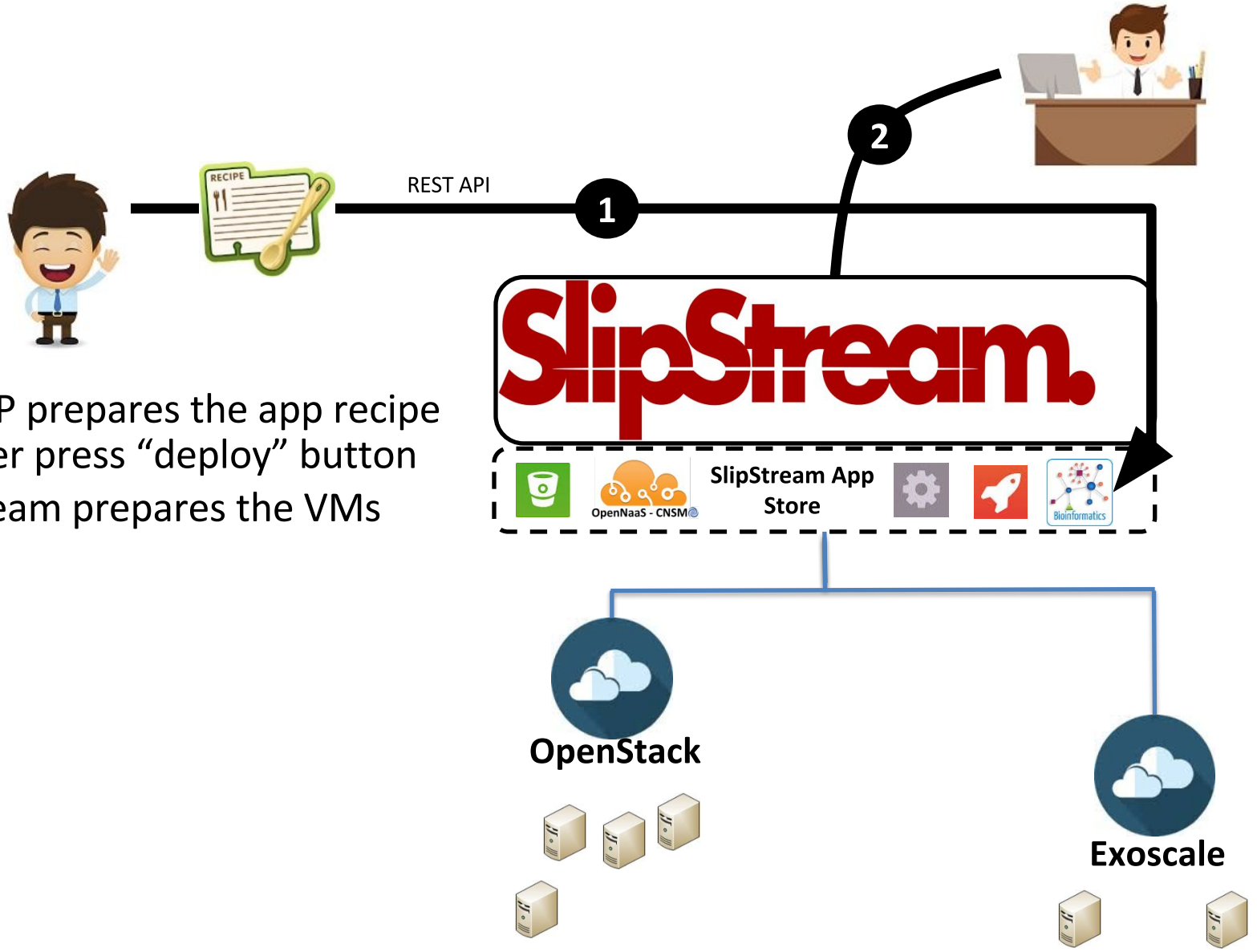
❶ The ASP prepares the app recipe

INTEGRATION OF NETWORK SERVICES (II)



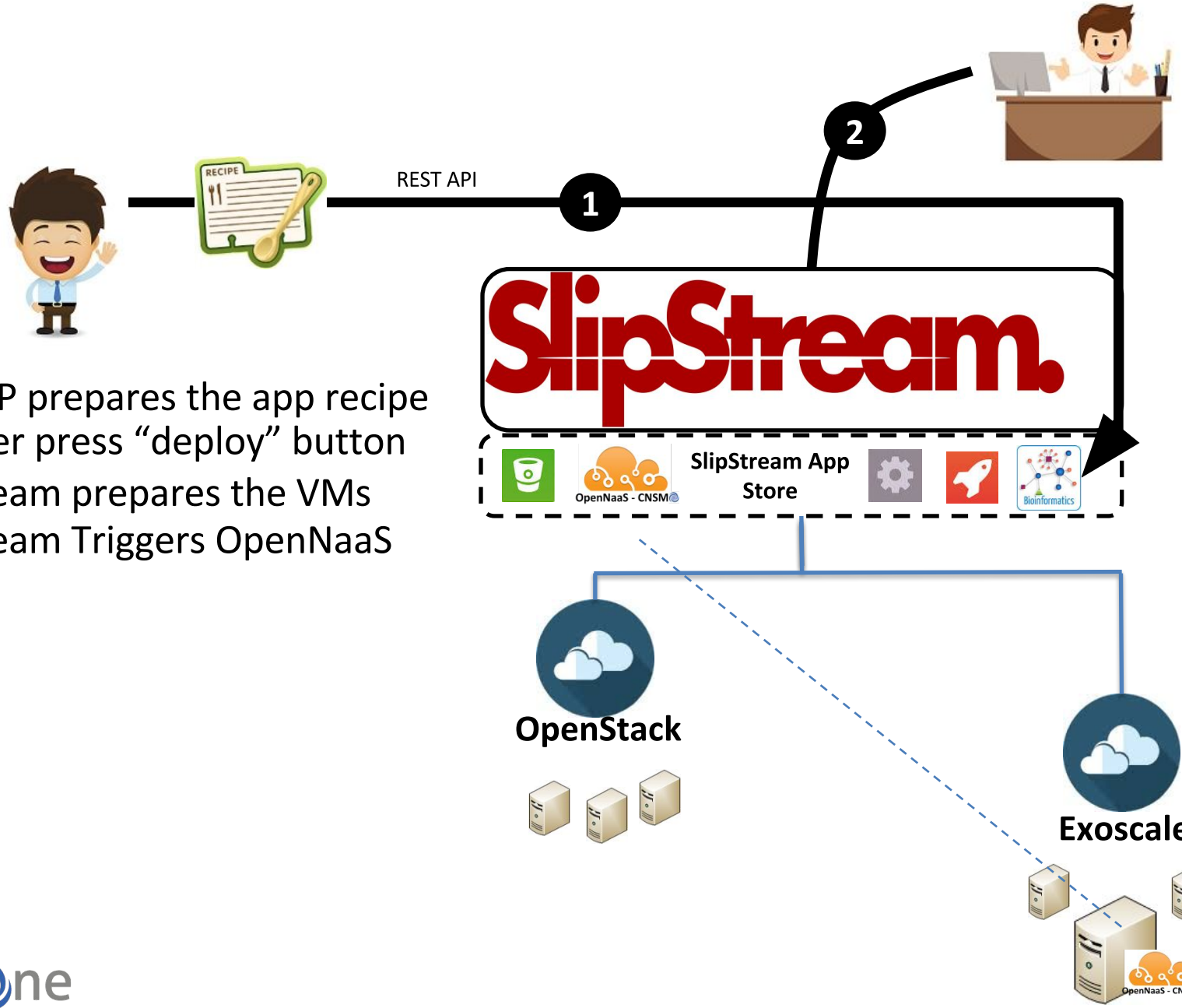
❶ The ASP prepares the app recipe
❷ The user press “deploy” button

INTEGRATION OF NETWORK SERVICES (III)



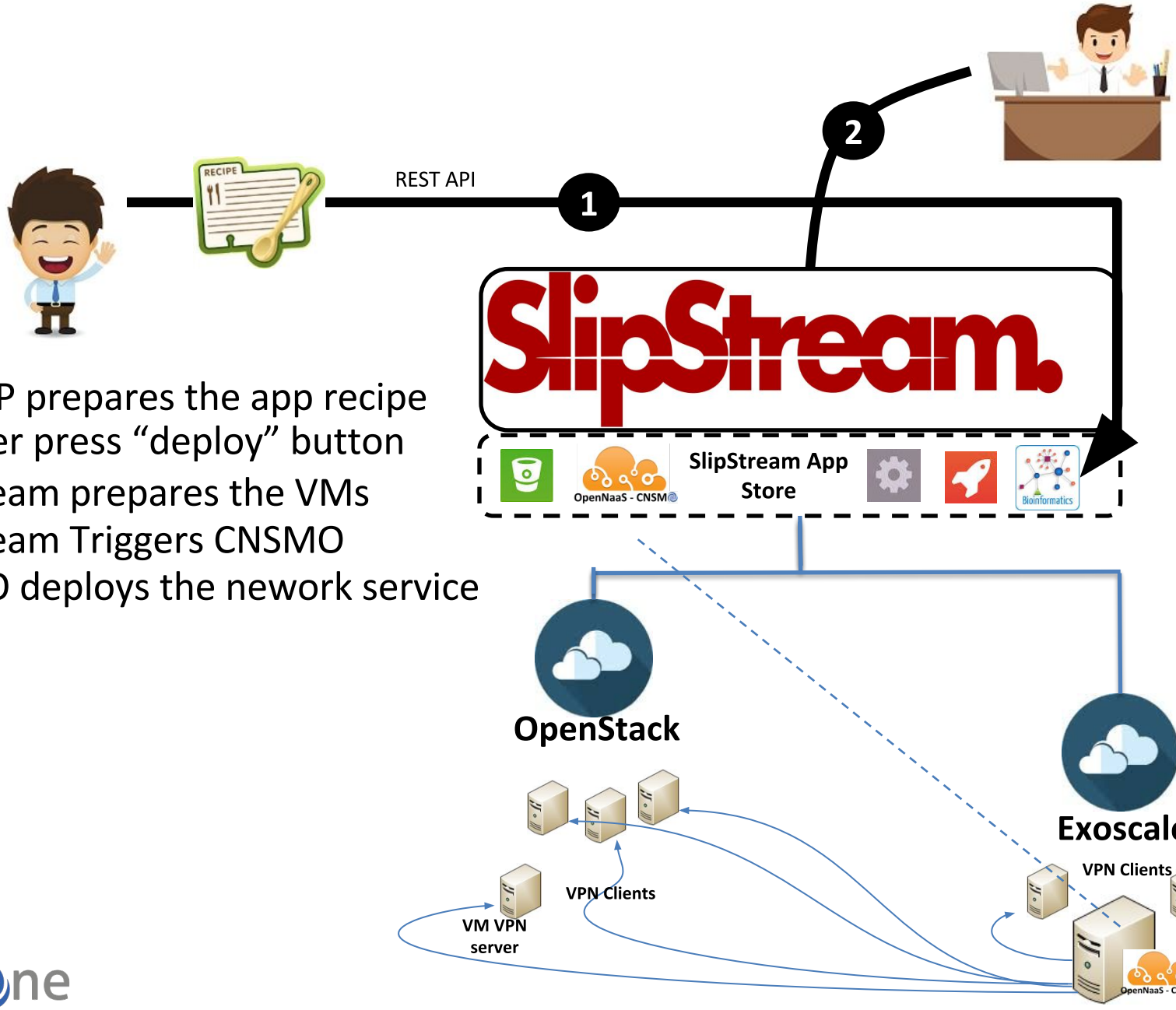
❶ The ASP prepares the app recipe
❷ The user press “deploy” button
❸ SlipStream prepares the VMs

INTEGRATION OF NETWORK SERVICES (IV)



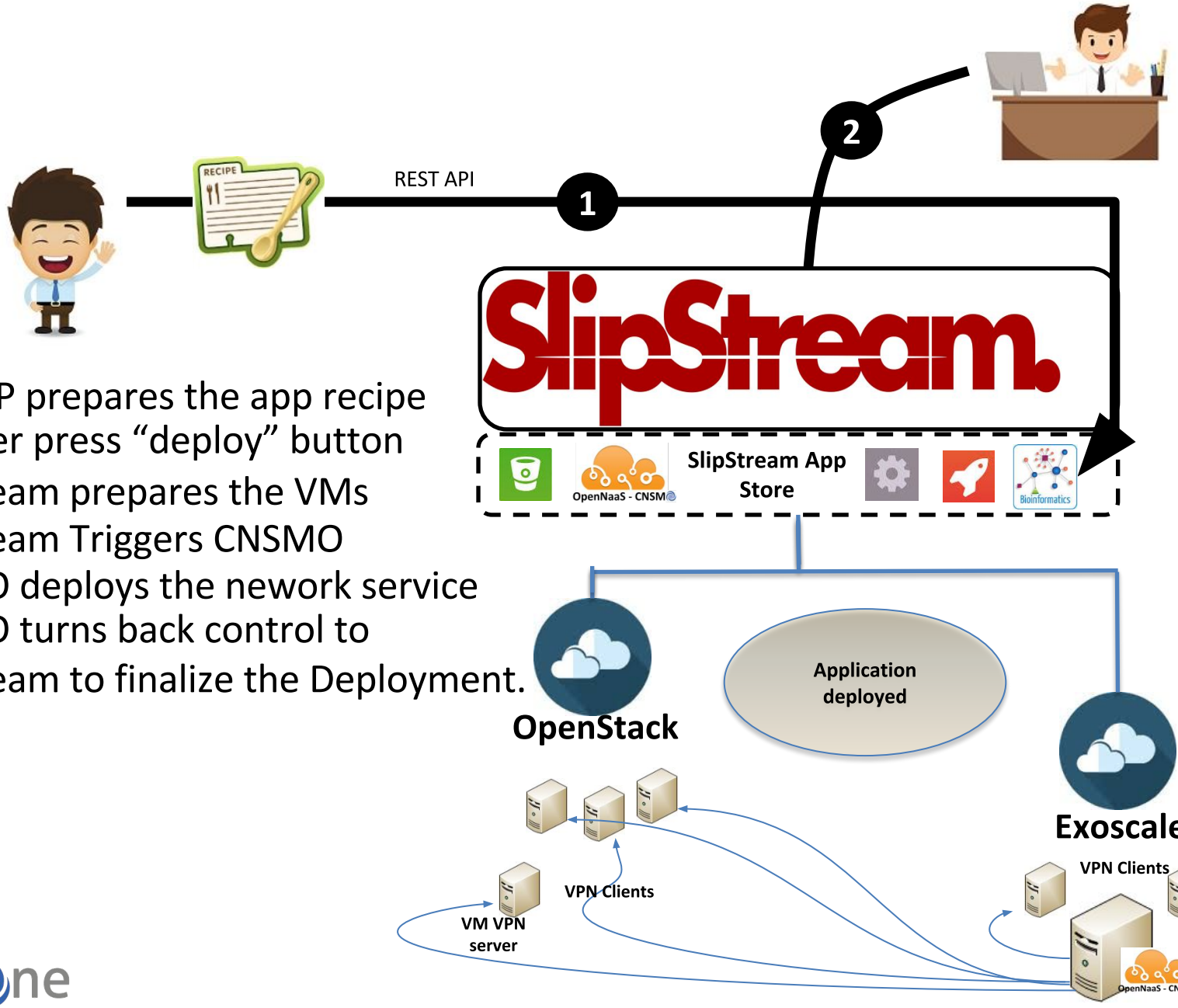
❶ The ASP prepares the app recipe
❷ The user press “deploy” button
❸ SlipStream prepares the VMs
❹ SlipStream Triggers OpenNaaS

INTEGRATION OF NETWORK SERVICES (V)



❶ The ASP prepares the app recipe
❷ The user press “deploy” button
❸ SlipStream prepares the VMs
❹ SlipStream Triggers CNSMO
❺ CNSMO deploys the nework service

INTEGRATION OF NETWORK SERVICES (VI)



❶ The ASP prepares the app recipe
❷ The user press “deploy” button
❸ SlipStream prepares the VMs
❹ SlipStream Triggers CNSMO
❺ CNSMO deploys the nework service
❻ CNSMO turns back control to SlipStream to finalize the Deployment.

Creating Network Services

1, 2, 3, **4**, 5, 6, 7

CNSMO SERVICES CREATION

- A CNSMOManager (agent) is required for each service. SlipStream application recipe is used to instantiate it.
- The service itself (server.py in sample below) may open a webserver to allow remote interaction, typically from other services

```
def get_server_app_request(host, port, service_id):
    bind_address = "0.0.0.0"
    d = dict(service_id=service_id,
             trigger='python server.py -a %s -p %s' % (bind_address, port),
resources=["https://raw.githubusercontent.com/dana-i2cat/cnsmo/develop/src/main/python/net/i2cat/cnsmoservices/fw/app/server.py",
                       "https://raw.githubusercontent.com/dana-i2cat/cnsmo-net-services/develop/src/main/docker/fw/Dockerfile",
                       "https://raw.githubusercontent.com/dana-i2cat/cnsmo-net-services/develop/src/main/docker/fw/sc-manager.py",],
             dependencies=[],
             endpoints=[{ "uri":"http://%s:%s/fw/" %(host, port), "driver":"REST", "logic":"post", "name":"add_rule"},
                        { "uri":"http://%s:%s/fw/" %(host, port), "driver":"REST", "logic":"delete", "name":"delete_rule"},])
    return d


def main(host, port, redis_address, service_id):
    bash_deployer = BashDeployer(None)
    server = CNSMOManager(redis_address, service_id, "FWServer", bash_deployer, None)
    server.start()
    time.sleep(0.5)
    server.compose_service(**get_server_app_request(host, port, service_id))
    server.launch_service(service_id)
```

CNSMO SERVICES CREATION

- Services may react to others being created by listening to the distributed state storage:

```
def __configure_system_state(self):
  self.__system_state_manager = SystemStateFactory.generate_system_state_client(self.__bind_address, "myVpn", "VPNManager",
                                                                                 self.__status, ["VPNServer", "VPNClient", "VPNConfigManager"],
                                                                                 self.register_service)
```

- CNSMO may create service proxies (local instance of a service bound to a remote service) leveraging the service metadata:

```
client_service = ServiceMaker().make_service("Client", self.__system_state_manager.load(service.get_service_id()).get_endpoints())
```

- See VPN manager source code here: https://github.com/dana-i2cat/cnsmo/blob/master/src/main/python/net/i2cat/cnsmoservices/vpn/manager/vpn.py

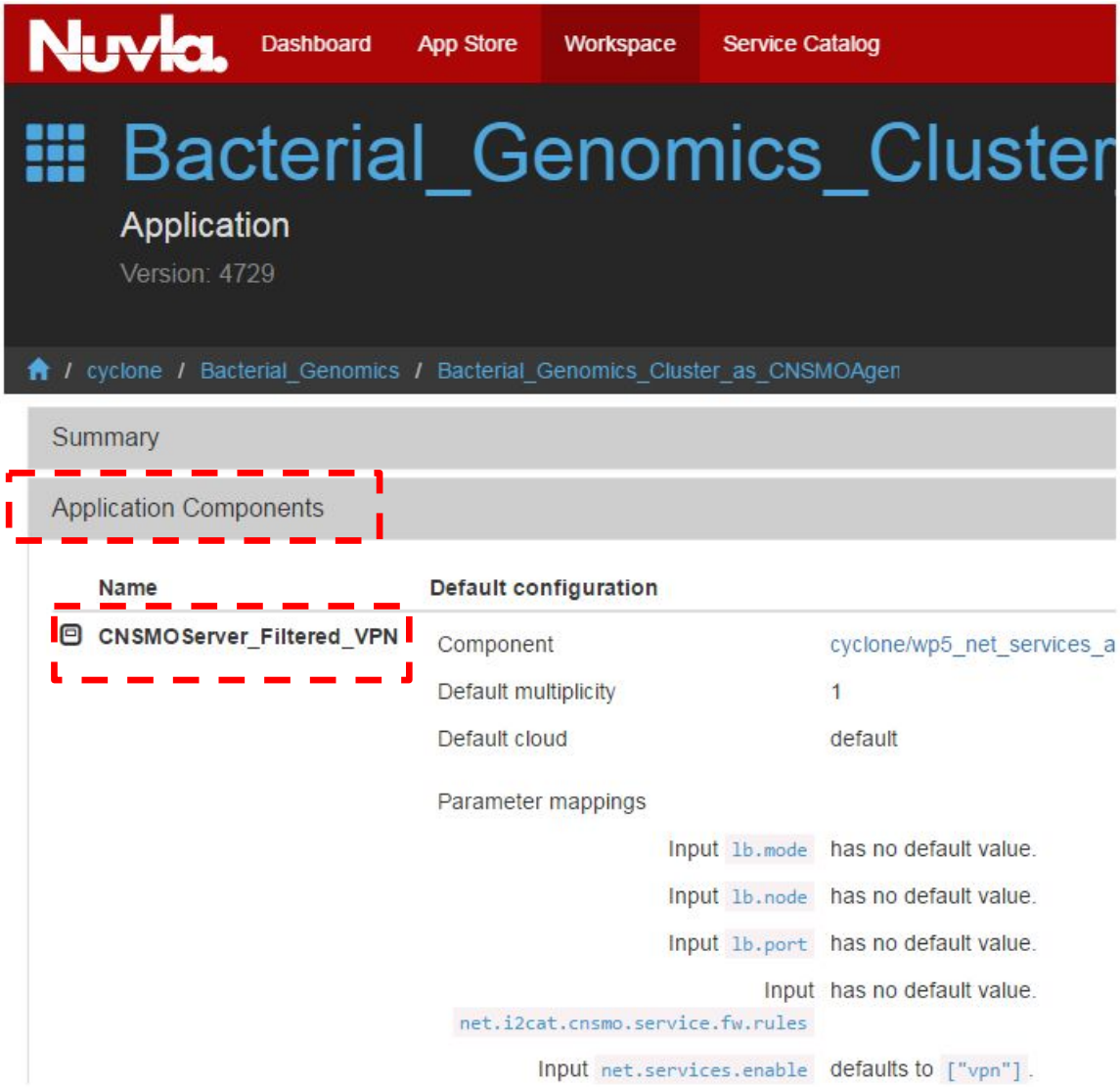
Integrating network services in your app

1, 2, 3, 4, **5**, 6, 7

INTEGRATION OF CNSMO SERVICES

- CNSMO requires that the components in the application are told to synchronize with it. The mechanism to do so is by means of SlipStream recipes.
- Integration check list:
  1. CNSMO server component (or a child of it) is part of the application
  2. For each component meant to run CNSMO services:
     - SlipStream recipes installs curl, git, cython, pythonpip before launching any service in this VM.
     - SlipStream recipes installs docker, downloads CNSMO and installs its requirements.
     - SlipStream recipes wait for CNSMO core to be ready,
     - SlipStream recipes launches an agent for each desired service.
     - For services that affect the interfaces available in the host (e.g. the VPN service), the application waits for the service to be ready before going further in the application configuration.

INTEGRATION OF CNSMO SERVICES



INTEGRATION OF CNSMO SERVICES

- CNSMO-enabled SlipStream application components available (based on ubuntu 14.04).
  - https://nuv.la/module/cyclone/wp5_net_services_applications/integrated/CNSMOServer
  - https://nuv.la/module/cyclone/wp5_net_services_applications/integrated/CNSMOAgent

- Inherit from them to be automatically integrated, OR
- Checkout their recipes and integrate them in your components

- Full integration guide here: http://opennaas.org/download/17172/

USING VPN SERVICE

- Required parameters:
  - CNSMO server:
    - Input **net.services.enable** to include **"vpn"**
  - CNSMO agents:
    - Input cnsmo.server.nodeinstanceid is mapped to output CNSMOServer:cnsmo.server.nodeinstanceid
    - Input vpn.server.nodeinstanceid is mapped to output CNSMOServer:vpn.server.nodeinstanceid.
    - Input **net.services.enable** to include **"vpn"**

- Provided parameters:
  - **vpn.address**: provides the IPv4 address of the interface in the VPN
  - **vpn.address6**: provides the IPv6 address of the interface in the VPN

  Use these parameters to configure your application to use the VPN

The service sets up a new logical interface for the VPN, using local openvpn installation.

USING FW SERVICE

- Required parameters:
  - CNSMO agents:
    - Input cnsmo.server.nodeinstanceid is mapped to output CNSMOServer:cnsmo.server.nodeinstanceid
    - Input **net.services.enable** to include **"fw"**
    - Input **net.i2cat.cnsmo.service.fw.rules** to include the JSON list of firewalling rules

- Provided parameters:
  None

This service is designed as a bootstrapping service, although the interface it opens remains accessible for further changes.

The service modifies local iptables configuration.

USING LB SERVICE

- Required parameters:
  - CNSMO server:
    - Input **net.services.enable** to include **"lb"**
    - Input **lb.mode** defaults to **leastconn.**
    - Input **lb.node** pointing to the **node name of the node to load balance**
    - Input **lb.port** defaults to **8080 (port to open)**
  - CNSMO agents:
    - Input cnsmo.server.nodeinstanceid is mapped to output CNSMOServer:cnsmo.server.nodeinstanceid
    - Input **net.services.enable** to include **"lb"**

WP5

Future Work

1, 2, 3, 4, 5, **6**, 7

FUTURE WORK (I)

- Additional network services to be compliant with UCs:

Item	Year 2 or Year 3	Result
DHCP Service	Year 2 and Year 3	DHCP service implemented and integrated as part of the CYCLONE solution.
DNS Service	Year 2 and Year 3	DNS service implemented and integrated as part of the CYCLONE solution.
Y2 use cases required services	Year 2 and Year 3	Potentially new demanded services that may arise while addressing Y2 use cases.

- Optimization of current services with additional functionalities:

Item	Year 2 or Year 3	Result
Scaling of Network services	Year 2	Scale up and down network services as required by the applications.
Resilience and fault tolerance	Year 3	Network services are recovered in case a general OpenNaaS CNSMO system takes place.
Secure services' bootstrap	Year 2	Network services deployment secure and trustable.
FW rules modification	Year 2 and Year 3	Users are able to on-demand re-configure pre-established FW rules.
FW improvement based on the SlipStream topology information.	Year 3	Starting from a topology of the application, the network is automatically configured.
Security groups improved granularity	Year 2 and Year 3	The network services can be deployed specifying more granular security groups than the ones offered by current CSP alternatives.

FUTURE WORK (II)

- Additional functionalities inherent to OpenNaaS-CNSMO:

Item	Year 2 or Year 3	Result
CNSMO monitoring tool	Year 2	API to visualize the current services launched by CNSMO and status
SDN/OF support	Year 2	Enable CNSMO with capability to control overlay OVS based networks
Integration with the security architecture	Year2 and Year 3	Extend CNSMO to integrate and provide logs to the login system.
Network services deployment without SlipStream	Year 2 and Year 3	CNSMO is independent of Cloud management tools as SlipStream.

OUTLINE

QA

1, 2, 3, 4, 5, 6, **7**

Cyclone

Q/A

THANKS FOR LISTENING

AND

HAVE A GOOD DAY

BACK UP SLIDES

WHAT CNSMO BRINGS TO CYCLONE: MAIN OBJECTIVES

- **OBJ2**: Improve cloud services by integrating network services into the cloud offering, allowing direct control over virtual machine network accessibility, intra-site data access, and inter-site data transfers.
  - **KPI-2.1**: Number of integrated Network functions (Target: 15, Achieved: xx).
  - **KPI-2.2**: Sum of KPI-2.2: Sum of story points for Network functions (Target: 45, Achieved: xx).

WHAT CNSMO BRINGS TO CYCLONE: MAIN OBJECTIVES

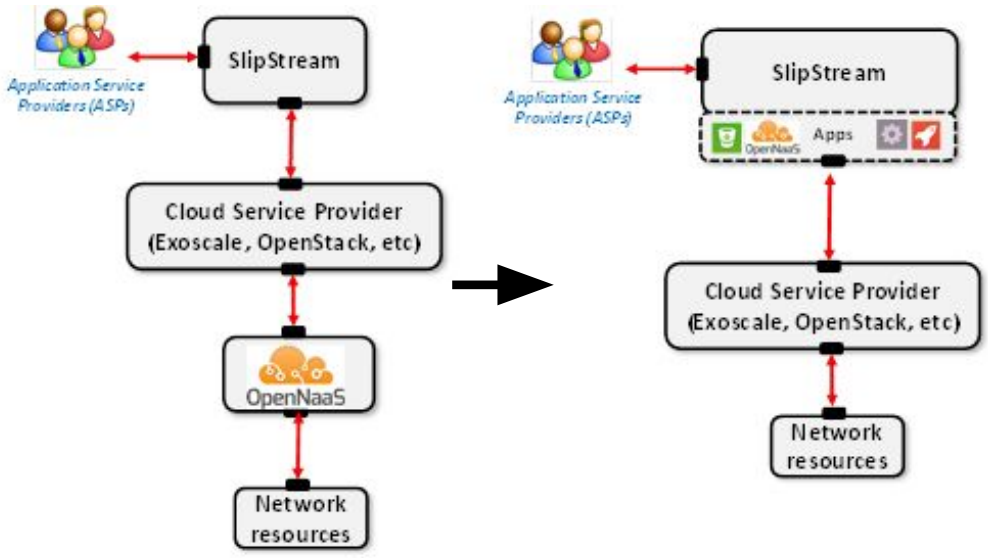
- **OBJ4**: Provide tools that allow application developers to take advantage of features like VM coordination within deployments, automated placement of service components, and scaling of service components, essentially providing them with the means to develop a Platform-as-a-Service (PaaS) offering.
  - **KPI-4.1:** Number of tooling functions, e.g., “deployment VM coordination”. (Target: 20, Achieved: xx)
  - **KPI-4.2:** Sum of story points of tooling functions (Target: 100, Achieved: xx).

LIMITATIONS

- List of objectives that with the current OpeNaaS cannot be achieved, because they don´t meake sense.
  - implement partitioning mechanisms and network abstracted information model for network resource composition purposes in other to expose management and control features to cloud federations
  - Isolation of network resources. This approach will improve the efficiency of the network infrastructure utilization.
  - Implementation of scheduling algorithms and mechanisms for logical networks composition driven from expertise obtained on the PHOSPHORUS FP6 EC and XIFI FP7.

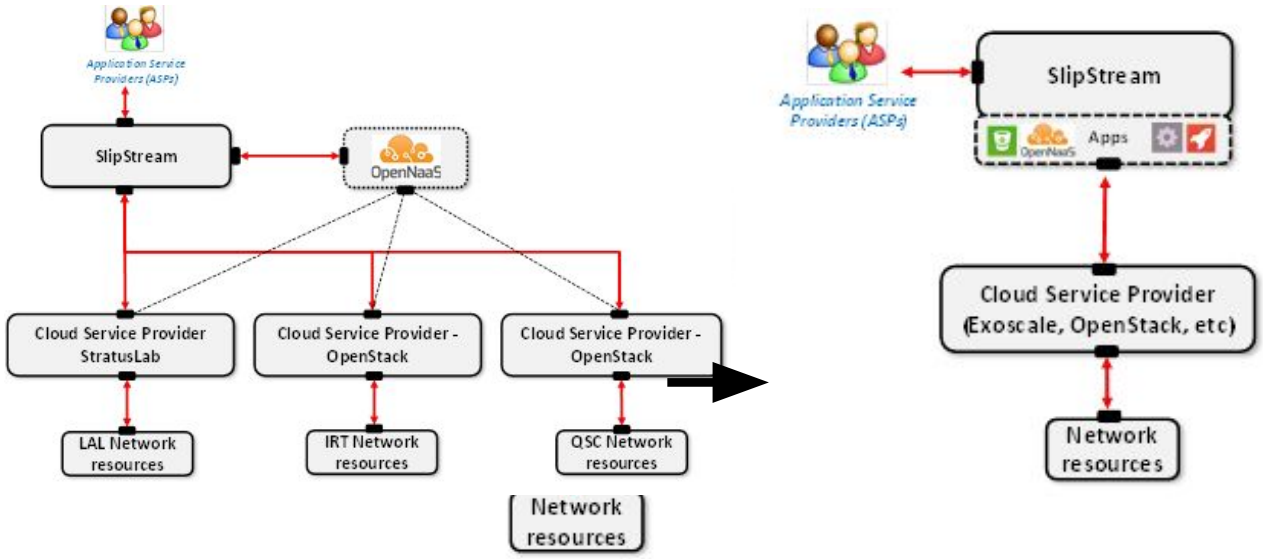
LIMITATIONS AND DECISIONS TAKEN (I)

- Access to infrastructure Resources:
  - Limitation: Market reality rules network service developments and integration.
  - Impact: Providers prevent from access to the physical infrastructure.
  - **Solution**: OpenNaaS CNSMO. Implement virtualized Network Functions (VNFs) to shape network micro-service



LIMITATIONS AND DECISIONS TAKEN (II)

- Integration with SlipStream.
  - Limitation: SlipStream cannot be dependent on a third party software (OpenNaaS).
  - Impact: Release network services as an extension of SlipStream is not a flexible solution
  - **Solution**: CYCLONE networking services are provided by application components, in the SlipStream app store.

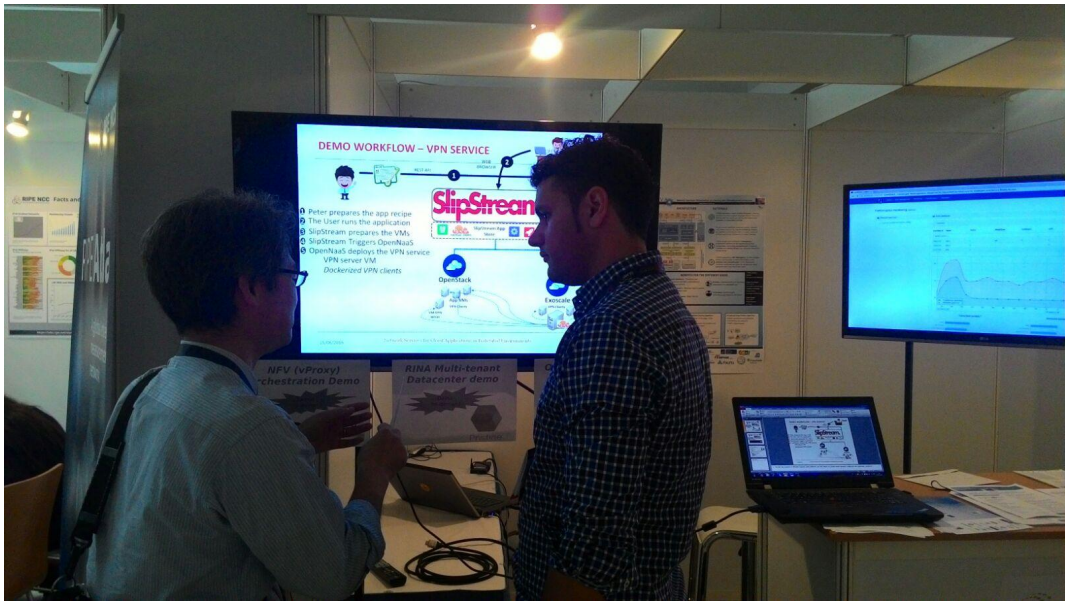


LIMITATIONS AND MITIGATION PLAN (II)

- Utilization and integration with Cloud Service Provider (CSP) tools.
  - Limitation: Integration with CSPs would be ad-hoc and constrained to their development.
  - Impact: It would entail an enormous effort to integrate with CSP platforms to provide network services.
  - **Solution**: OpenNaaS CNSMO. implement Virtualized Network Functions (VNFs) to shape network micro-services.

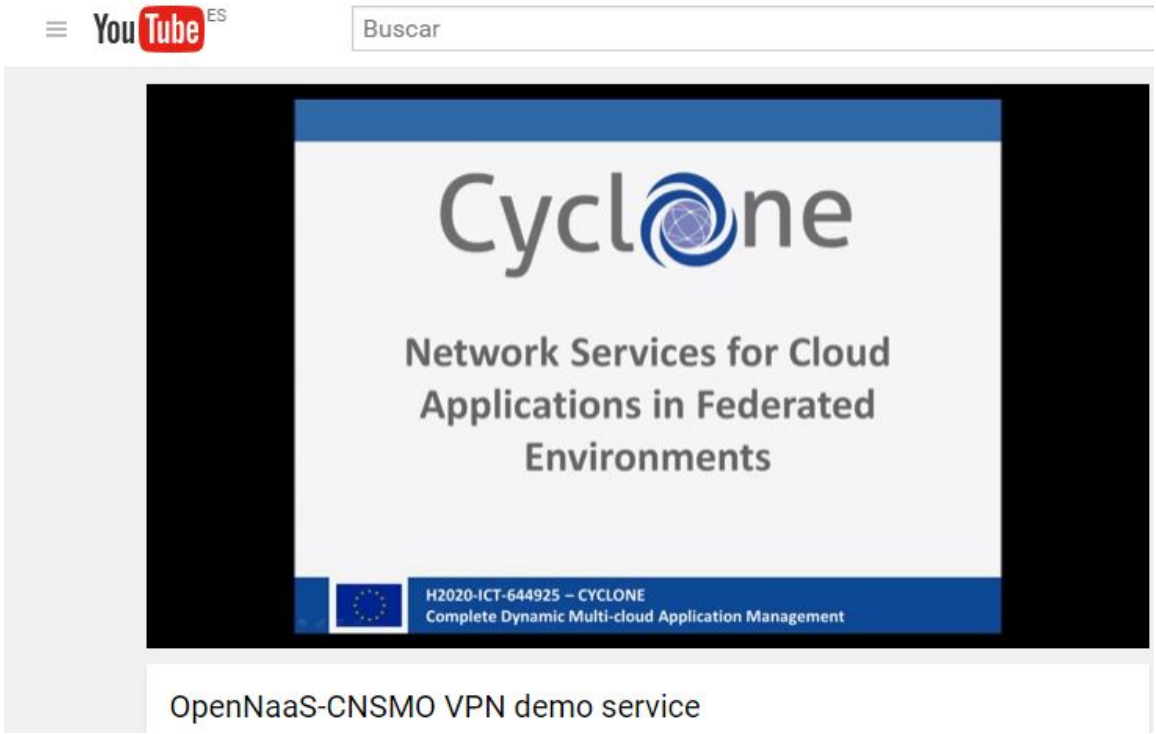
WP5: TASK 5.3 – DEMONSTRATION AND REPORTING

- Demonstration of CNSMO and network services.
  - Med Hoc Net conference
  - NetFutures'16
  - Cloud challenges meetings
  - TNC'16



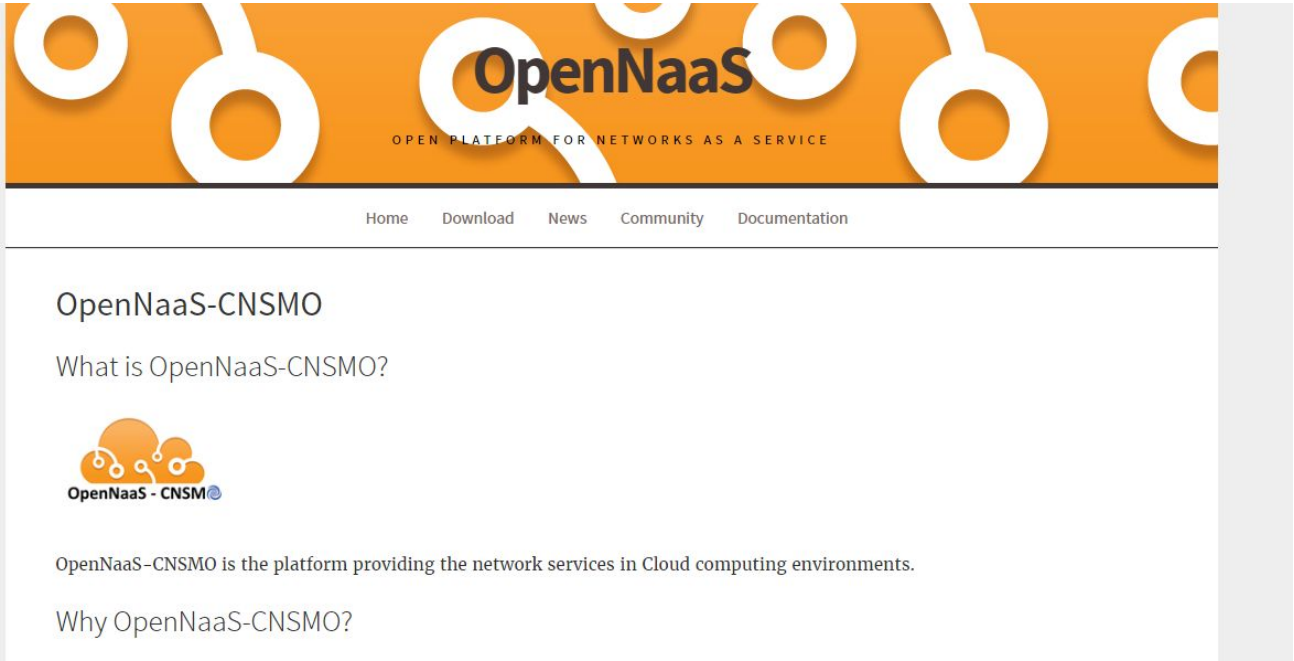
WP5: TASK 5.3 – DEMONSTRATION AND REPORTING

- Reporting
  - Youtube videos recorded.
    - VPN service: https://www.youtube.com/watch?v=34nqiouZly4
    - Firewall Service: https://www.youtube.com/watch?v=e7JcFwJz-bA
    - Load Balancer service: https://www.youtube.com/watch?v=IpPkZAAfqo8



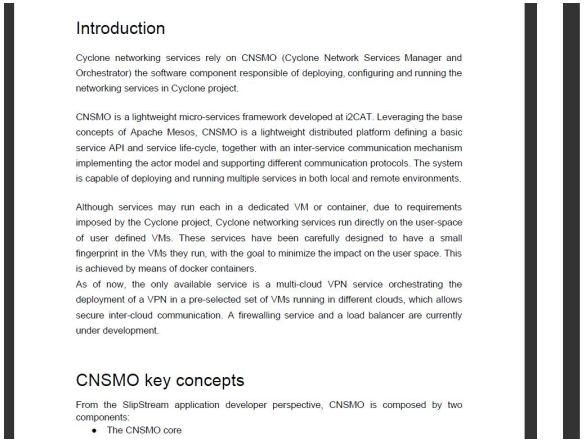
WP5: TASK 5.3 – DEMONSTRATION AND REPORTING

- Reporting
  - OpenNaaS Website



WP5: TASK 5.3 – DEMONSTRATION AND REPORTING

- Documentation:



Introduction

Cyclone networking services rely on CNSMO (Cyclone Network Services Manager and Orchestrator) the software component responsible of deploying, configuring and running the networking services in Cyclone project.

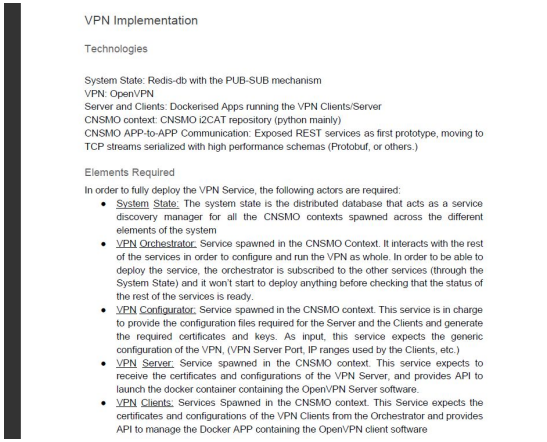
CNSMO is a lightweight micro-services framework developed at i2CAT. Leveraging the base concepts of Apache Mesos, CNSMO is a lightweight distributed platform defining a basic service API and service life-cycle, together with an inter-service communication mechanism implementing the actor model and supporting different communication protocols. The system is capable of deploying and running multiple services in both local and remote environments.

Although services may run each in a dedicated VM or container, due to requirements imposed by the Cyclone project, Cyclone networking services run directly on the user-space of user defined VMs. These services have been carefully designed to have a small fingerprint in the VMs they run, with the goal to minimize the impact on the user space. This is achieved by means of docker containers.
As of now, the only available service is a multi-cloud VPN service orchestrating the deployment of a VPN in a pre-selected set of VMs running in different clouds, which allows secure inter-cloud communication. A firewalling service and a load balancer are currently under development.

CNSMO key concepts

From the SlipStream application developer perspective, CNSMO is composed by two components:

- The CNSMO core



VPN Implementation

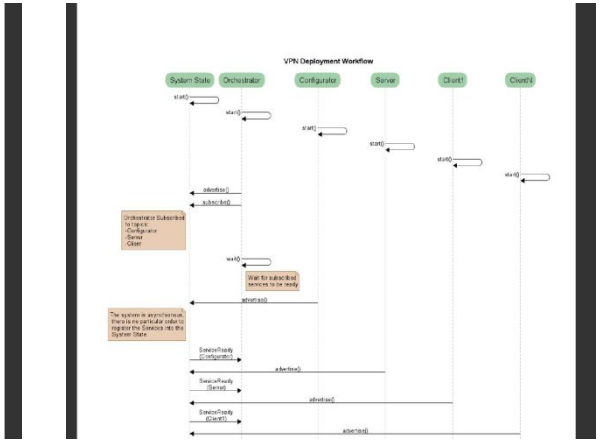
Technologies

System State: Redis-db with the PUB-SUB mechanism
VPN: OpenVPN
Server and Clients: Dockerised Apps running the VPN Clients/Server
CNSMO context: CNSMO i2CAT repository (python mainly)
CNSMO APP-to-APP Communication: Exposed REST services as first prototype, moving to TCP streams serialized with high performance schemas (Protobuf, or others.)

Elements Required

In order to fully deploy the VPN Service, the following actors are required:

- System State: The system state is the distributed database that acts as a service discovery manager for all the CNSMO contexts spawned across the different elements of the system
- VPN Orchestrator: Service spawned in the CNSMO Context. It interacts with the rest of the services in order to configure and run the VPN as whole. In order to be able to deploy the service, the orchestrator is subscribed to the other services (through the System State) and it won't start to deploy anything before checking that the status of the rest of the services is ready.
- VPN Configurator: Service spawned in the CNSMO context. This service is in charge to provide the configuration files required for the Server and the Clients and generate the required certificates and keys. As input, this service expects the generic configuration of the VPN, (VPN Server Port, IP ranges used by the Clients, etc.)
- VPN Server: Service spawned in the CNSMO context. This service expects to receive the certificates and configurations of the VPN Server, and provides API to launch the docker container containing the OpenVPN Server software.
- VPN Clients: Services Spawned in the CNSMO context. This Service expects the certificates and configurations of the VPN Clients from the Orchestrator and provides API to manage the Docker APP containing the OpenVPN client software



- Code available
  - github.com/dana-i2cat/cnsmo