Open**Stack**

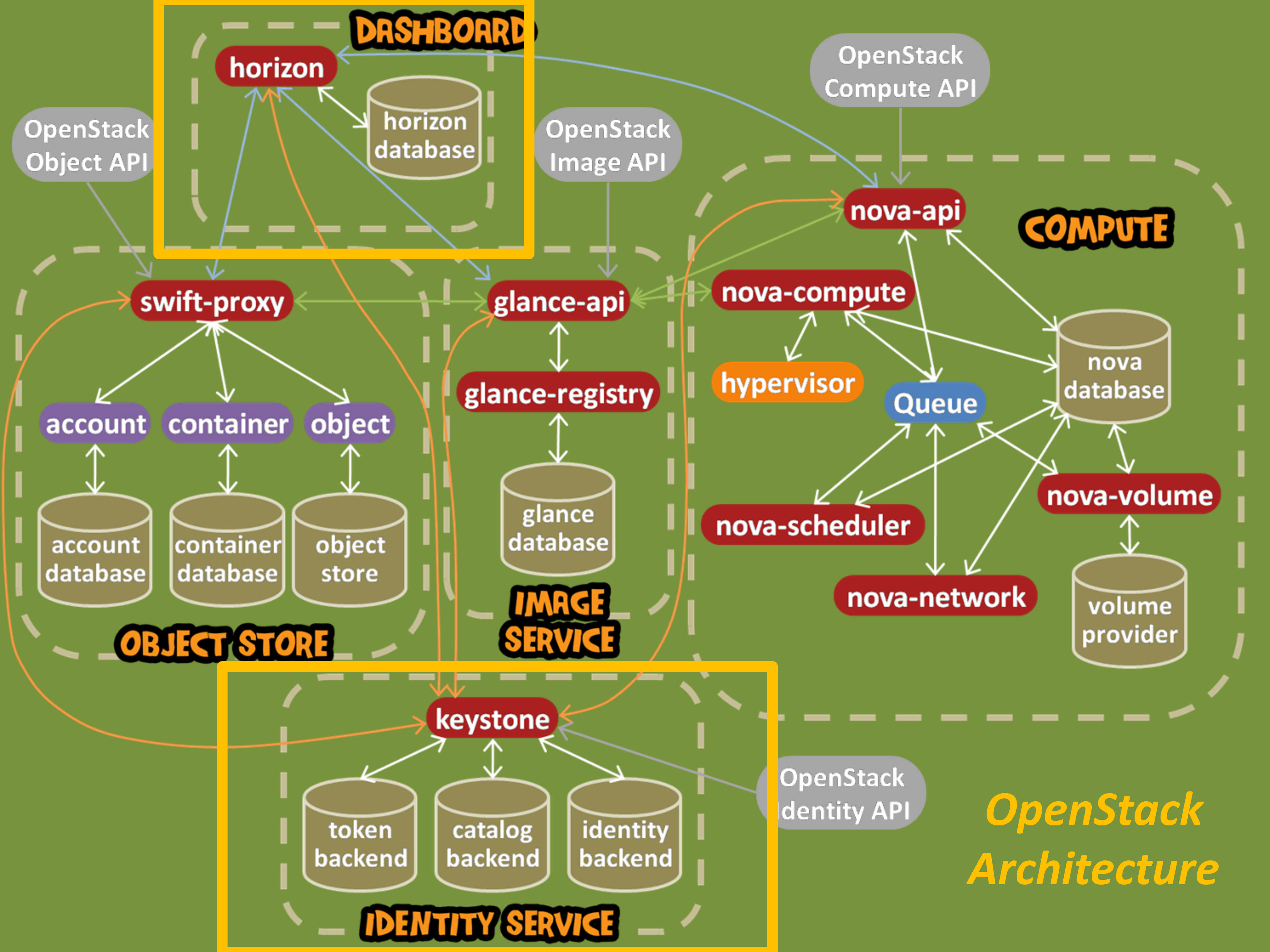
Architecture and Operation

OpenStack Tutorial Day 2

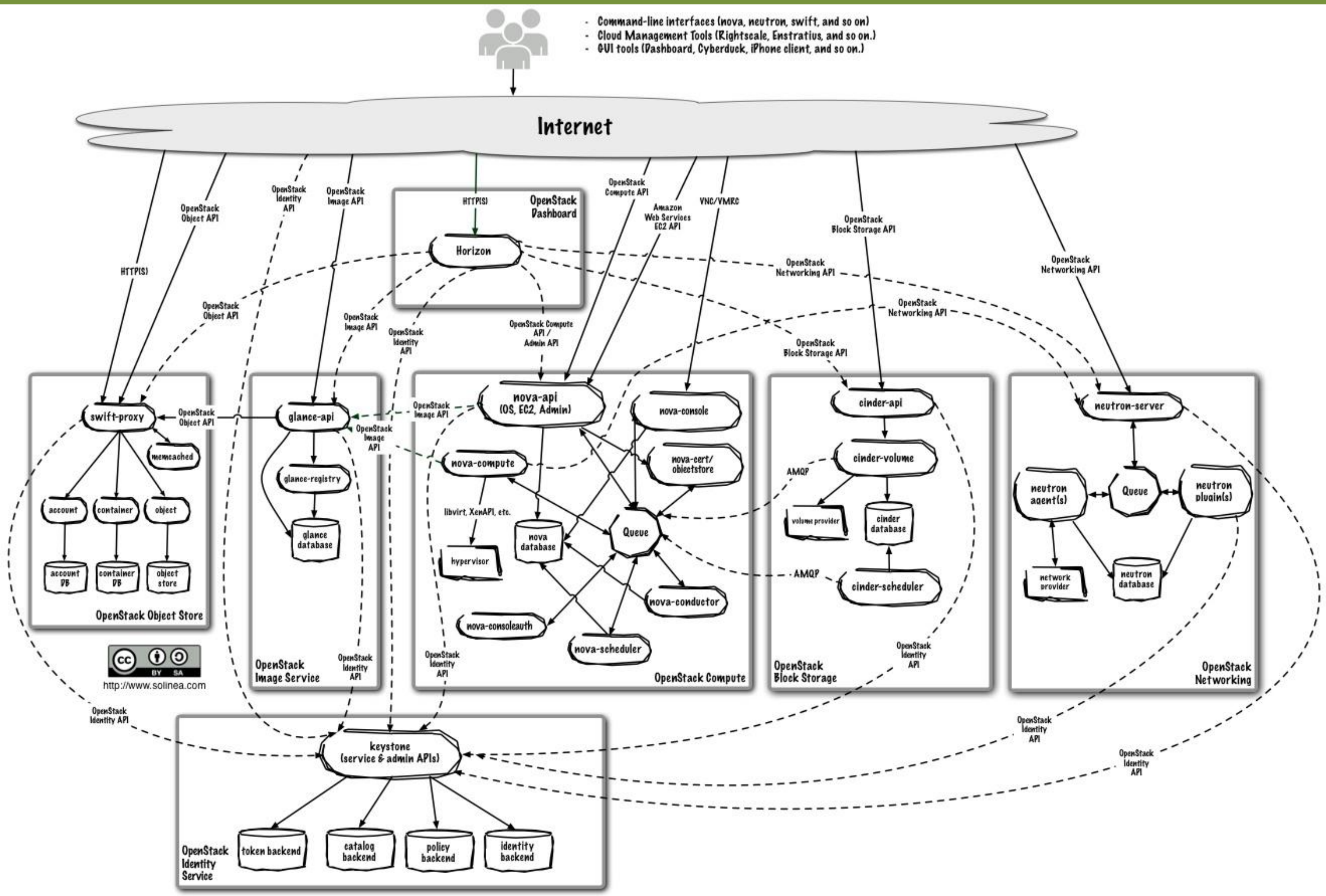
Kasidit Chanchio
Vasabilab,
Thammasat University

How OpenStack Components work

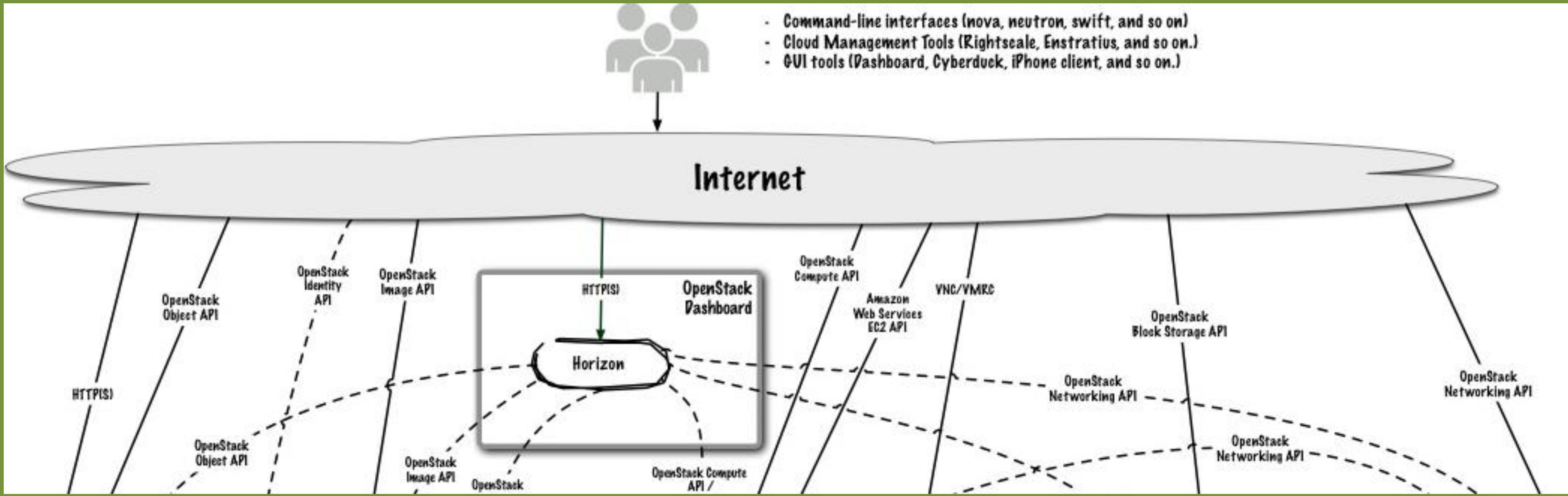
- Components in our focuses:
  - Keystone
  - Nova
  - Glance
  - Networking
  - Orchrestration


DASHBOARD
horizon
horizon database
OpenStack Object API
OpenStack Image API
OpenStack Compute API
nova-api
COMPUTE
swift-proxy
glance-api
nova-compute
hypervisor
Queue
nova database
glance-registry
account
container
object
account database
container database
object store
glance database
nova-volume
nova-scheduler
nova-network
volume provider
OBJECT STORE
IMAGE SERVICE
keystone
token backend
catalog backend
identity backend
OpenStack Identity API
IDENTITY SERVICE
OpenStack Architecture

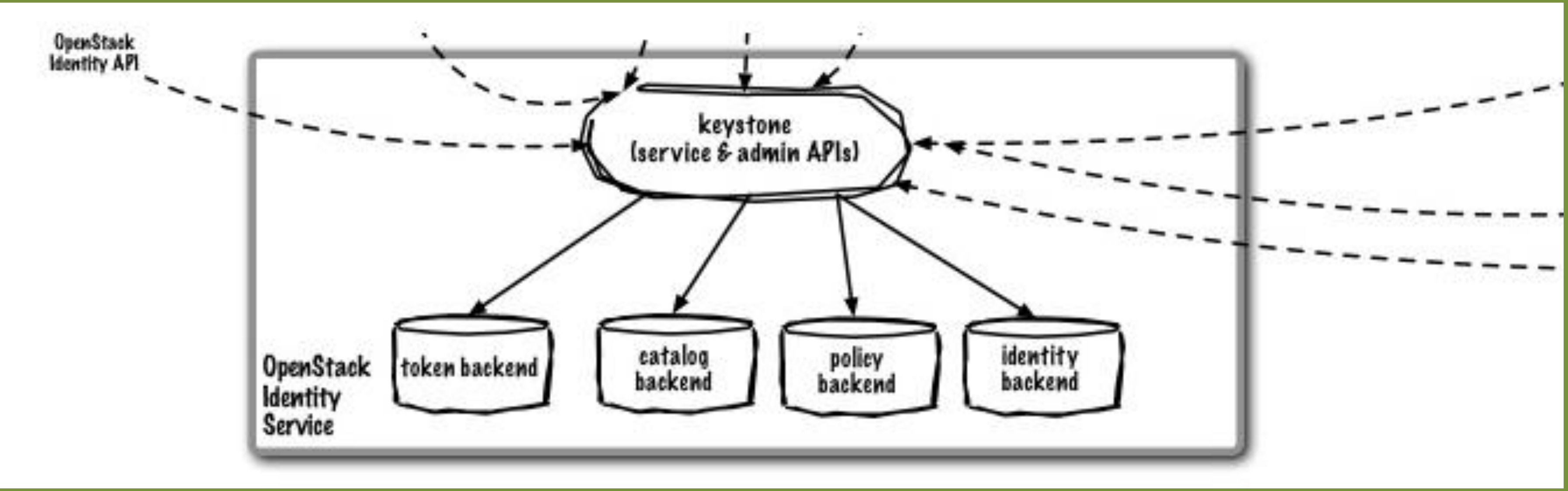
OpenStack Architecture



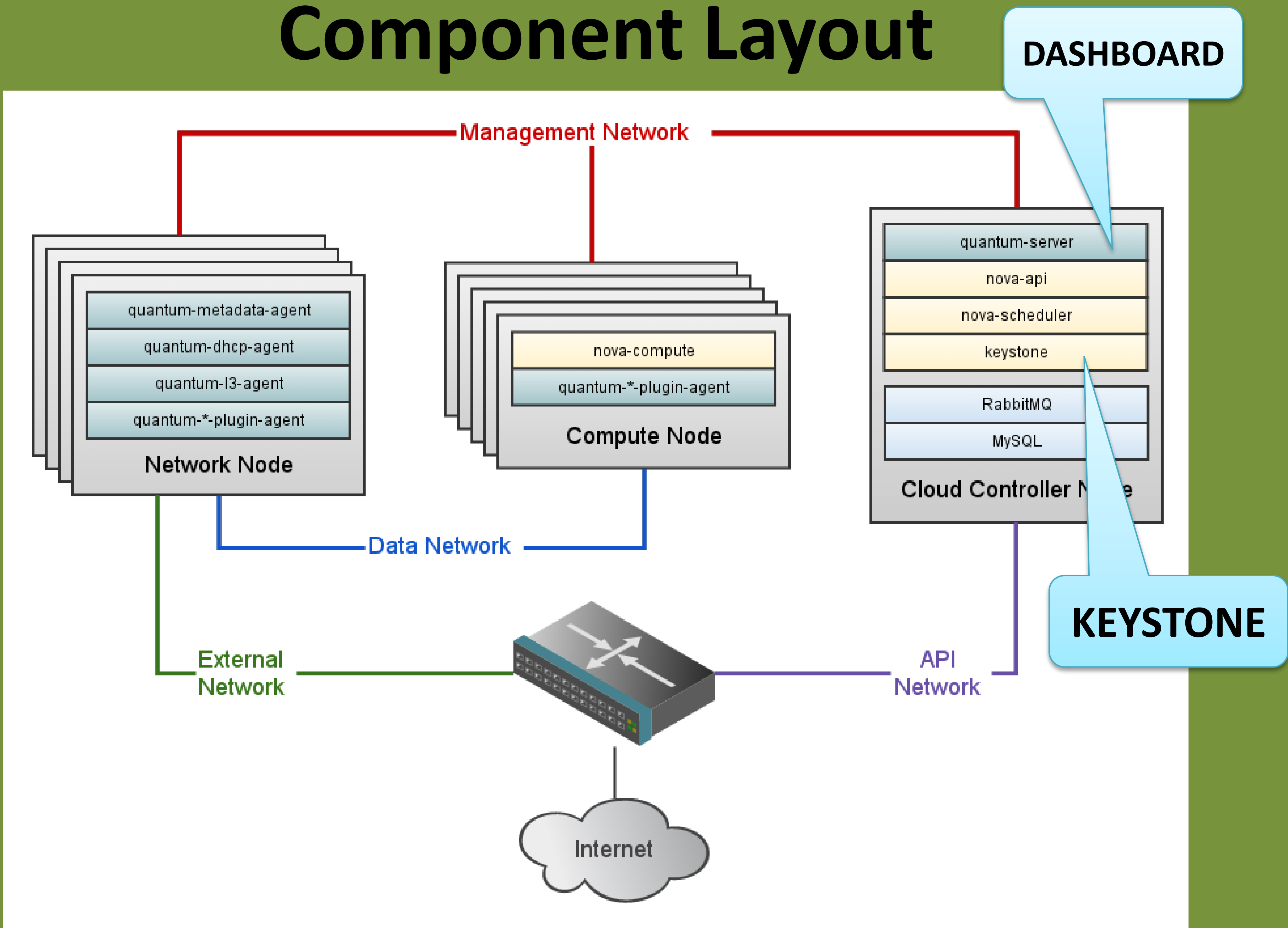
OpenStack Architecture



OpenStack Architecture



Component Layout



Keystone

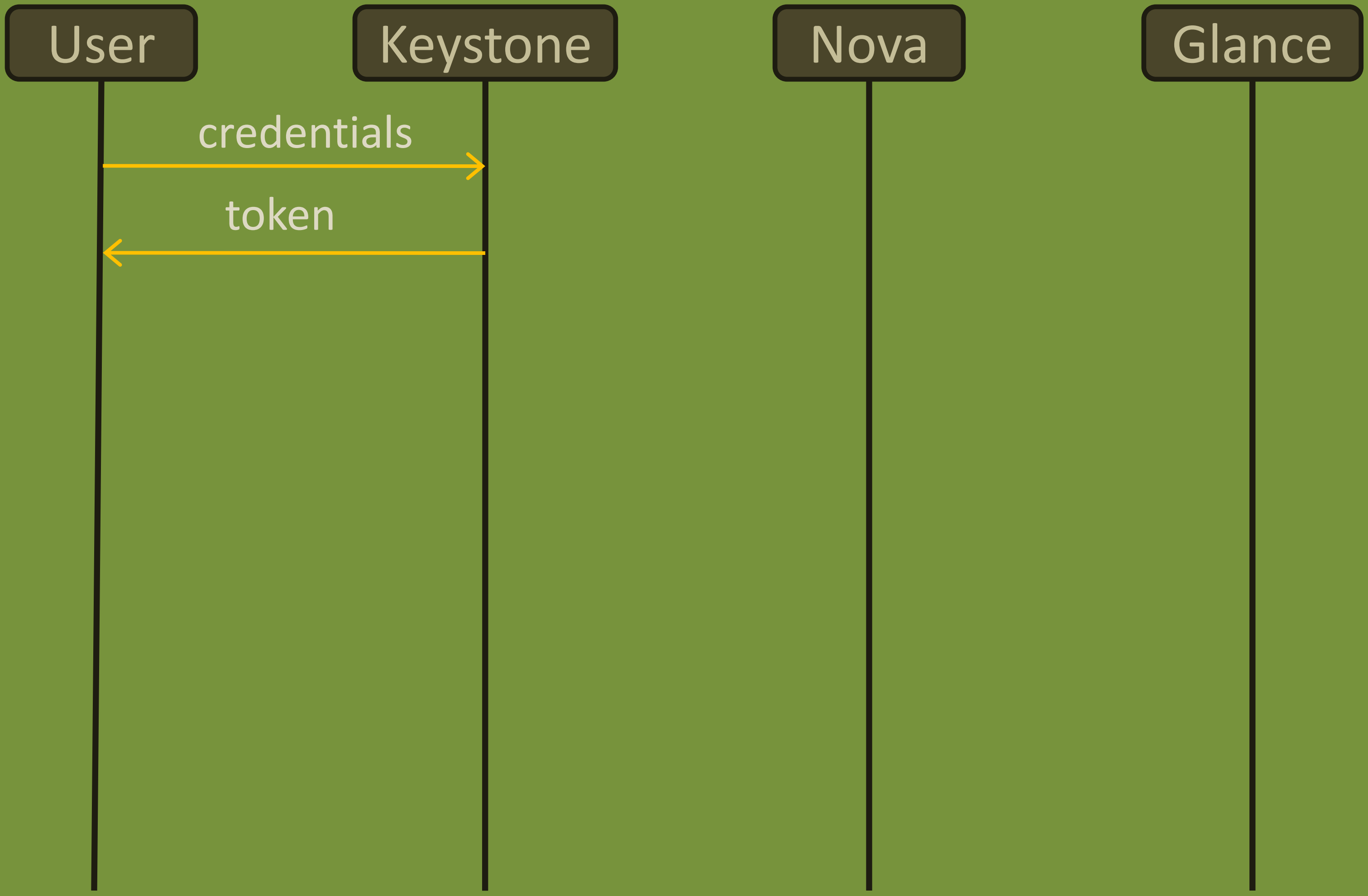
- A central authentication and authorization
- User represents someone or something that can gain access through Keystone. Users come with credentials that can be checked like passwords or API keys.
- Tenant represents what is called the project in Nova. Users are bound to a tenant by assigning them a role on that tenant.
- Role represents a number of privileges or rights a user has or actions they are allowed to perform.
- To access a service, we have to know its endpoint. So there are endpoint templates in Keystone that provide information about all existing endpoints of all existing services.

Keystone

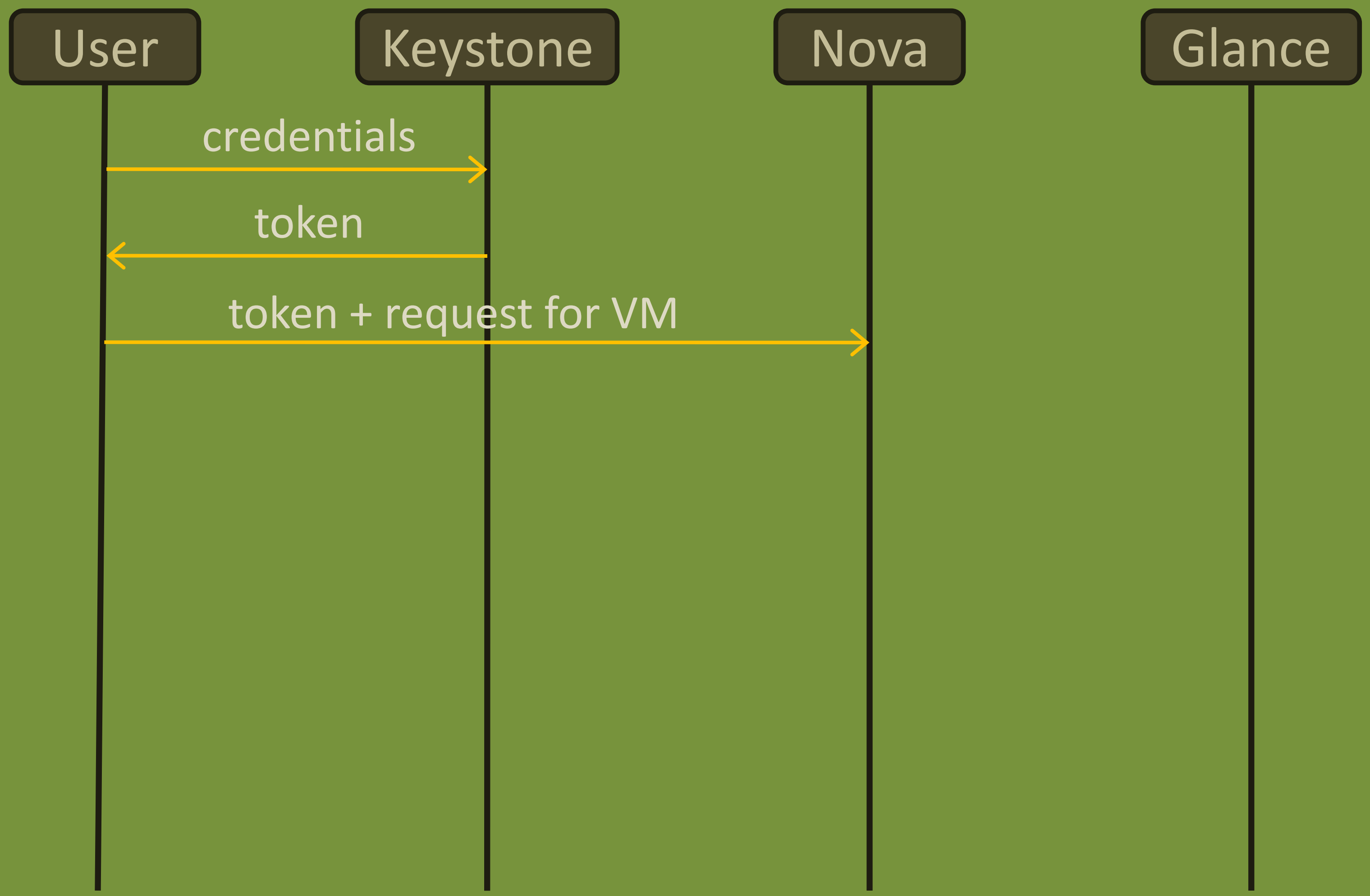
- To access some service, users provide their credentials to Keystone and receive a token.
- If the user, for example, wants to spawn a new VM instance in Nova, one can find an URL to Nova in the list of endpoints provided by Keystone and send an appropriate request.
- After that, Nova verifies the validity of the token in Keystone and should create an instance from some image by the provided image ID and plug it into some network.
- All the way this token travels between services so that they can ask Keystone or each other for additional information or some actions.

Keystone Control Flow

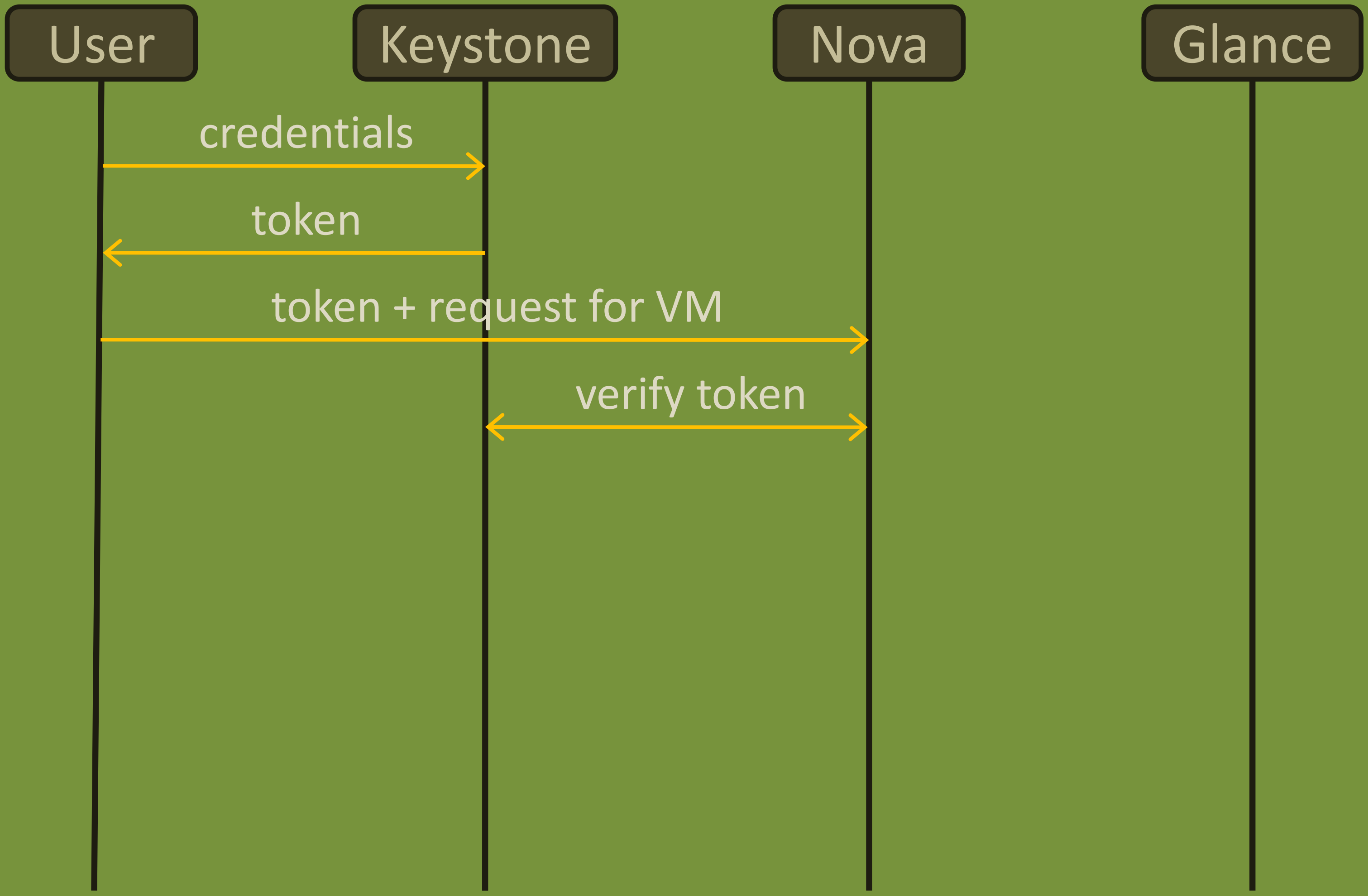
Keystone Control Flow



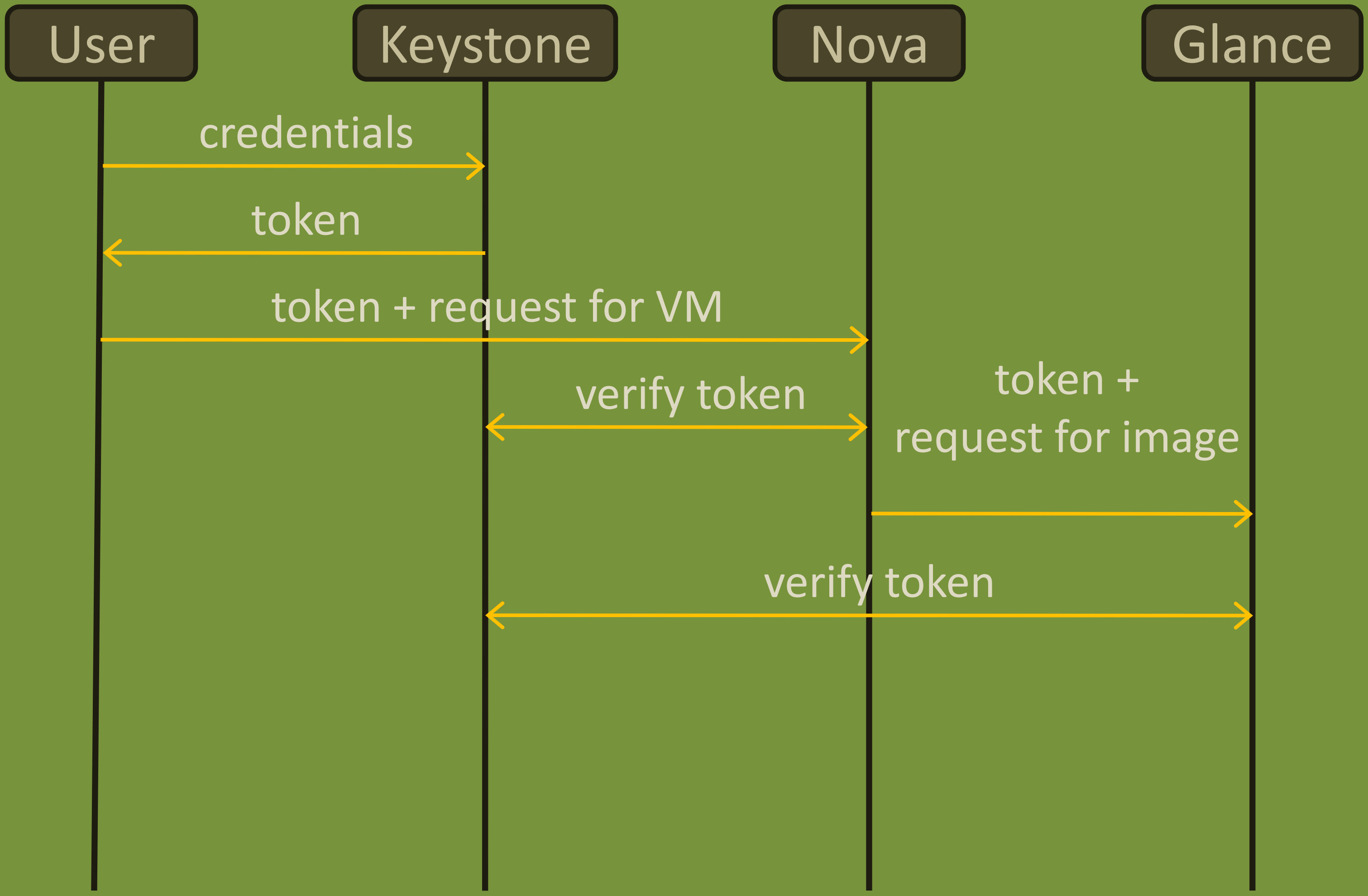
Keystone Control Flow



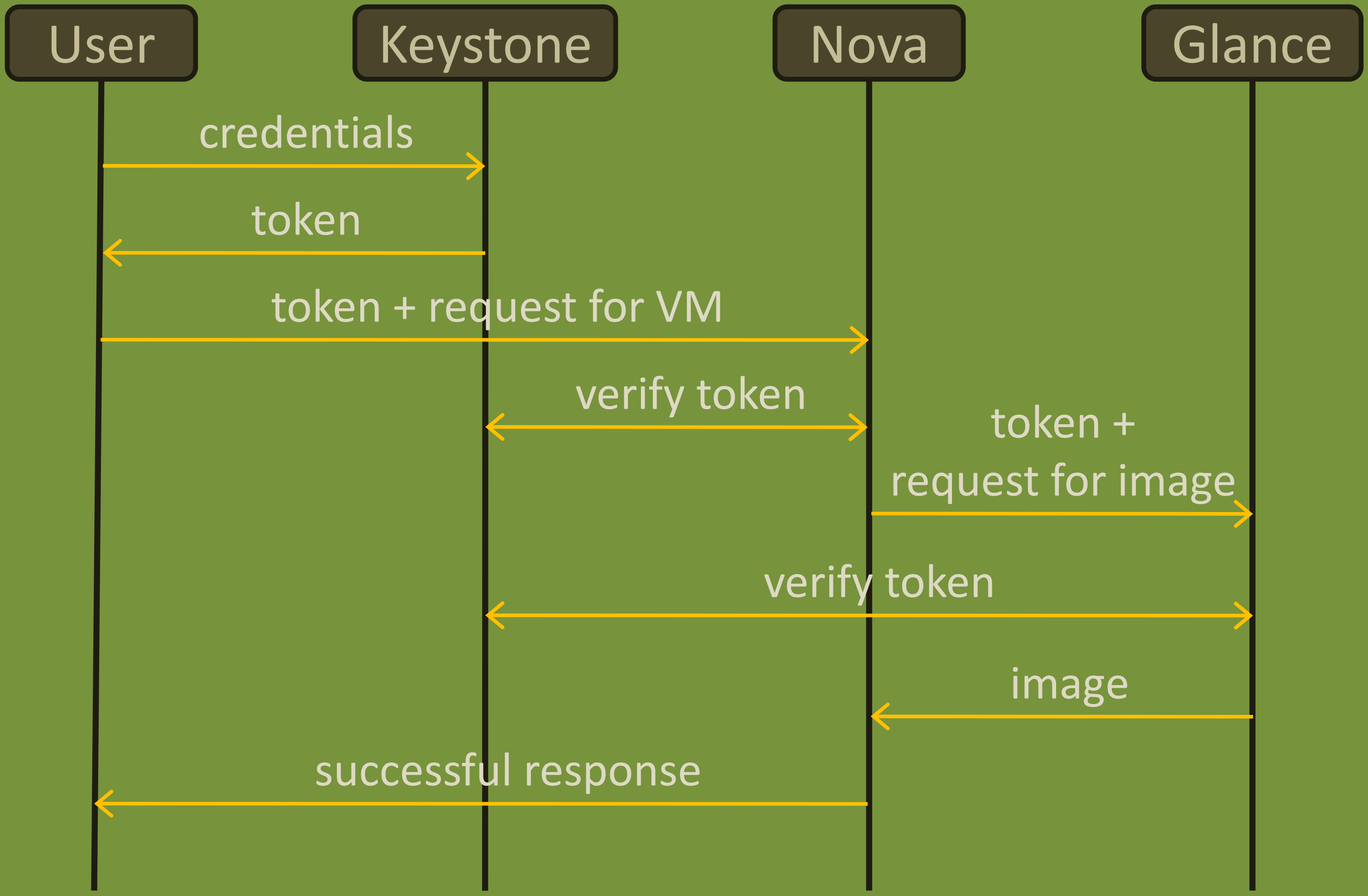
Keystone Control Flow

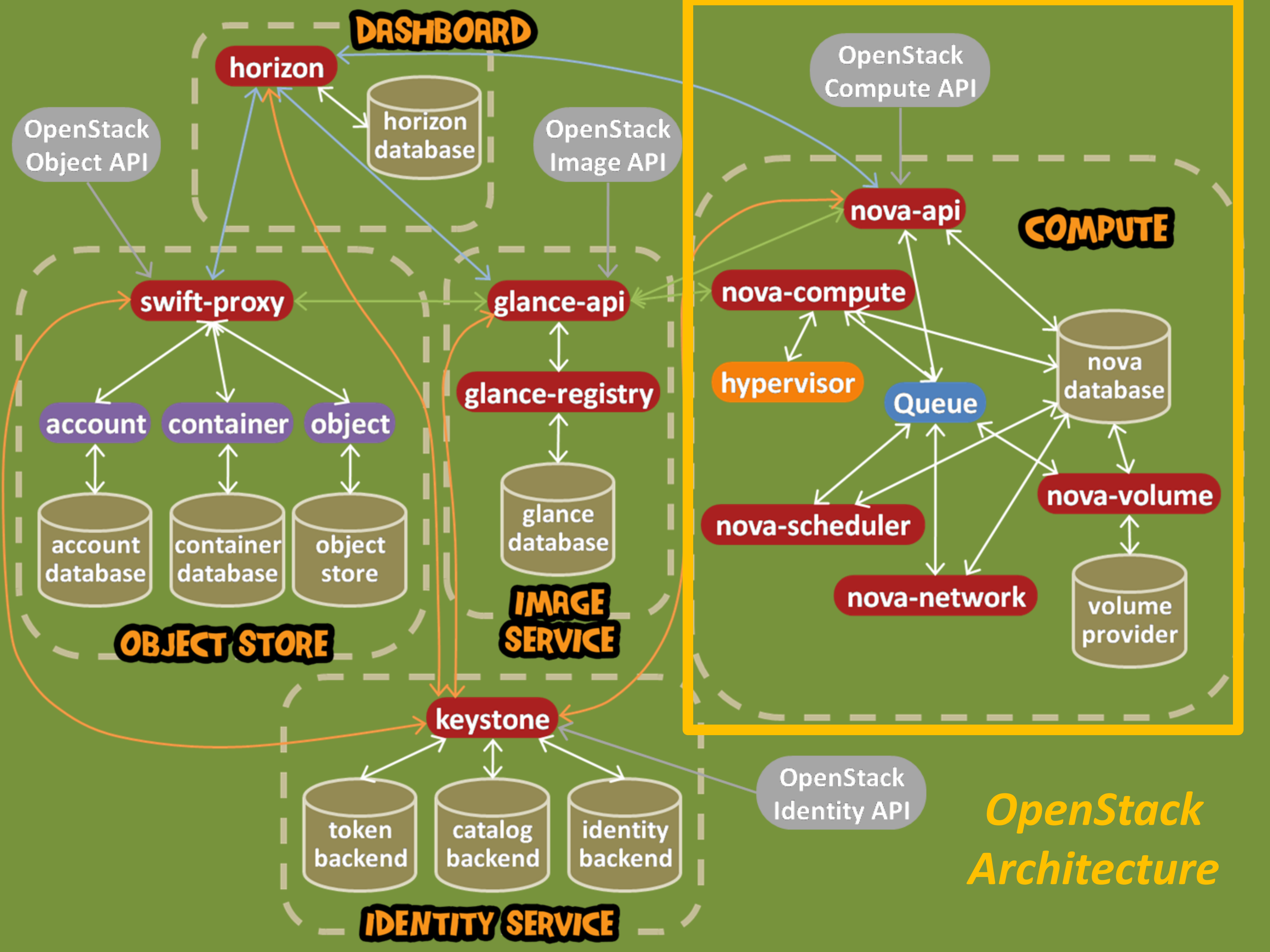


Keystone Control Flow

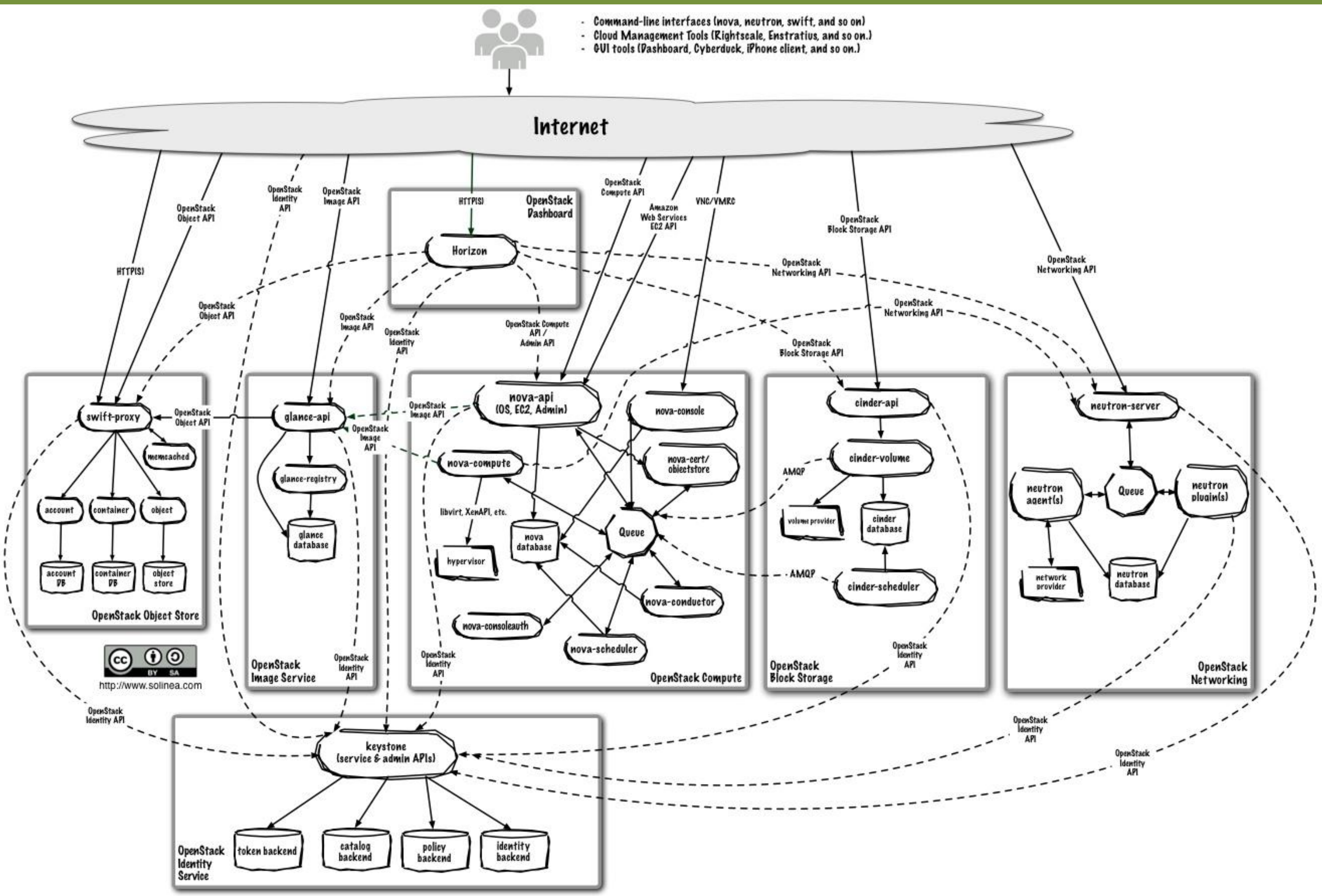


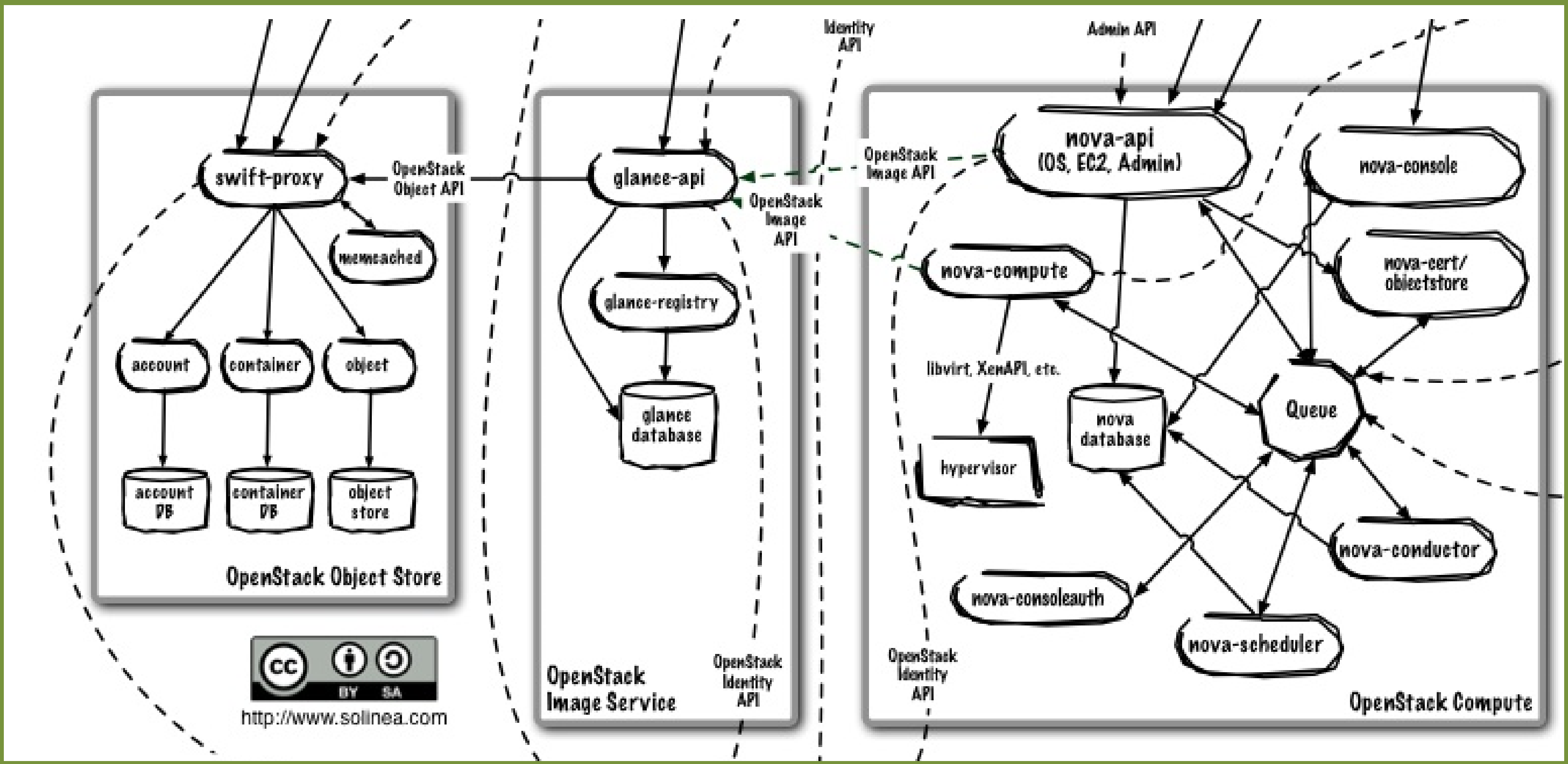
Keystone Control Flow




DASHBOARD
horizon
horizon database
OpenStack Object API
OpenStack Image API
OpenStack Compute API
nova-api
COMPUTE
swift-proxy
glance-api
nova-compute
hypervisor
Queue
nova database
glance-registry
account
container
object
account database
container database
object store
glance database
nova-volume
nova-scheduler
nova-network
volume provider
OBJECT STORE
IMAGE SERVICE
keystone
token backend
catalog backend
identity backend
OpenStack Identity API
IDENTITY SERVICE
OpenStack Architecture

OpenStack Architecture



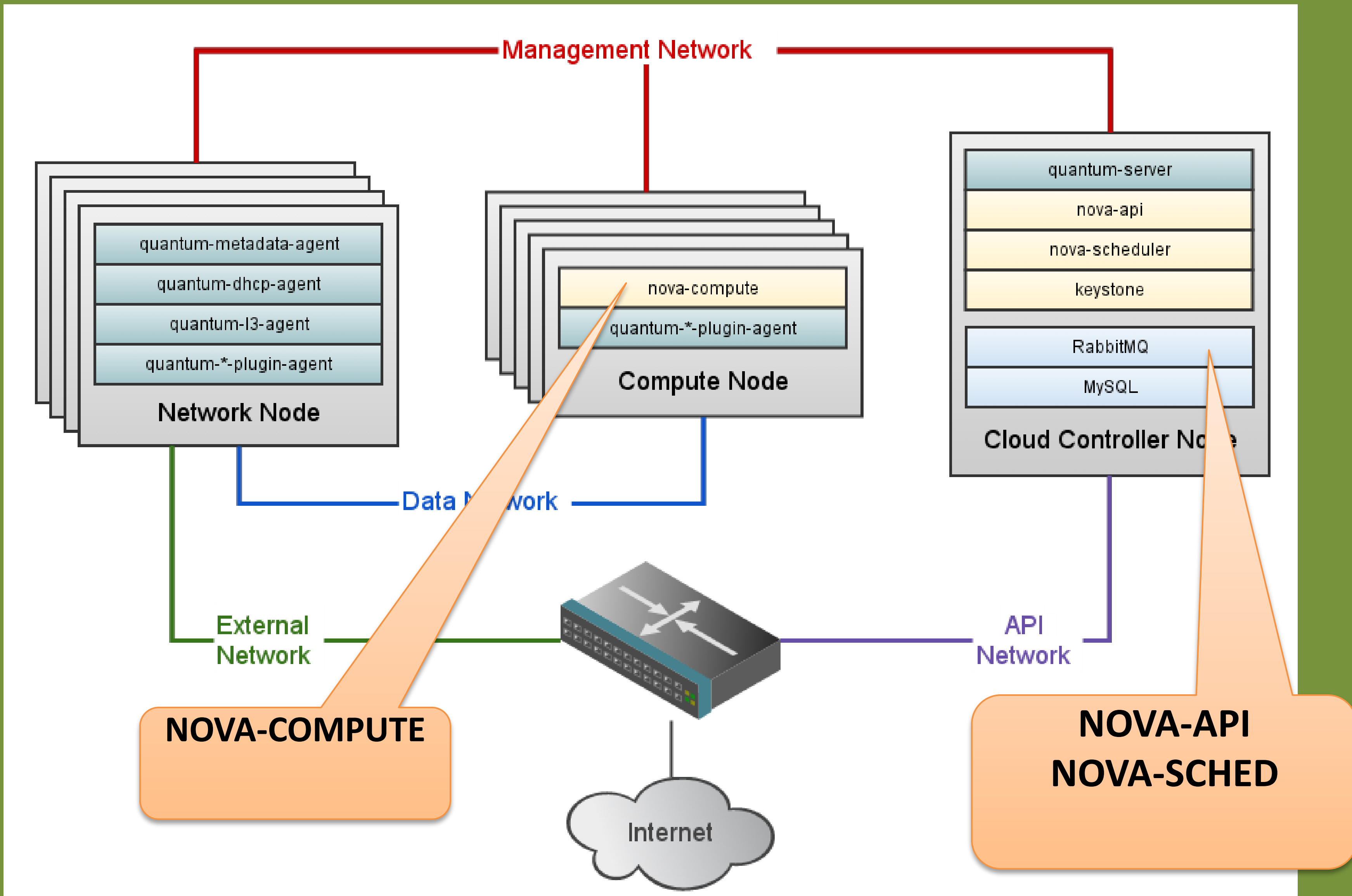


Identity API
Admin API
swift-proxy
OpenStack Object API
memcached
account
container
object
account DB
container DB
object store
OpenStack Object Store
glance-api
glance-registry
glance database
OpenStack Image Service
OpenStack Identity API
OpenStack Image API
OpenStack Image API
nova-api (OS, EC2, Admin)
nova-console
nova-cert/ objectstore
nova-compute
libvirt, XenAPI, etc.
hypervisor
nova database
Queue
nova-conductor
nova-consoleauth
nova-scheduler
OpenStack Identity API
OpenStack Compute
BY SA
http://www.solinea.com

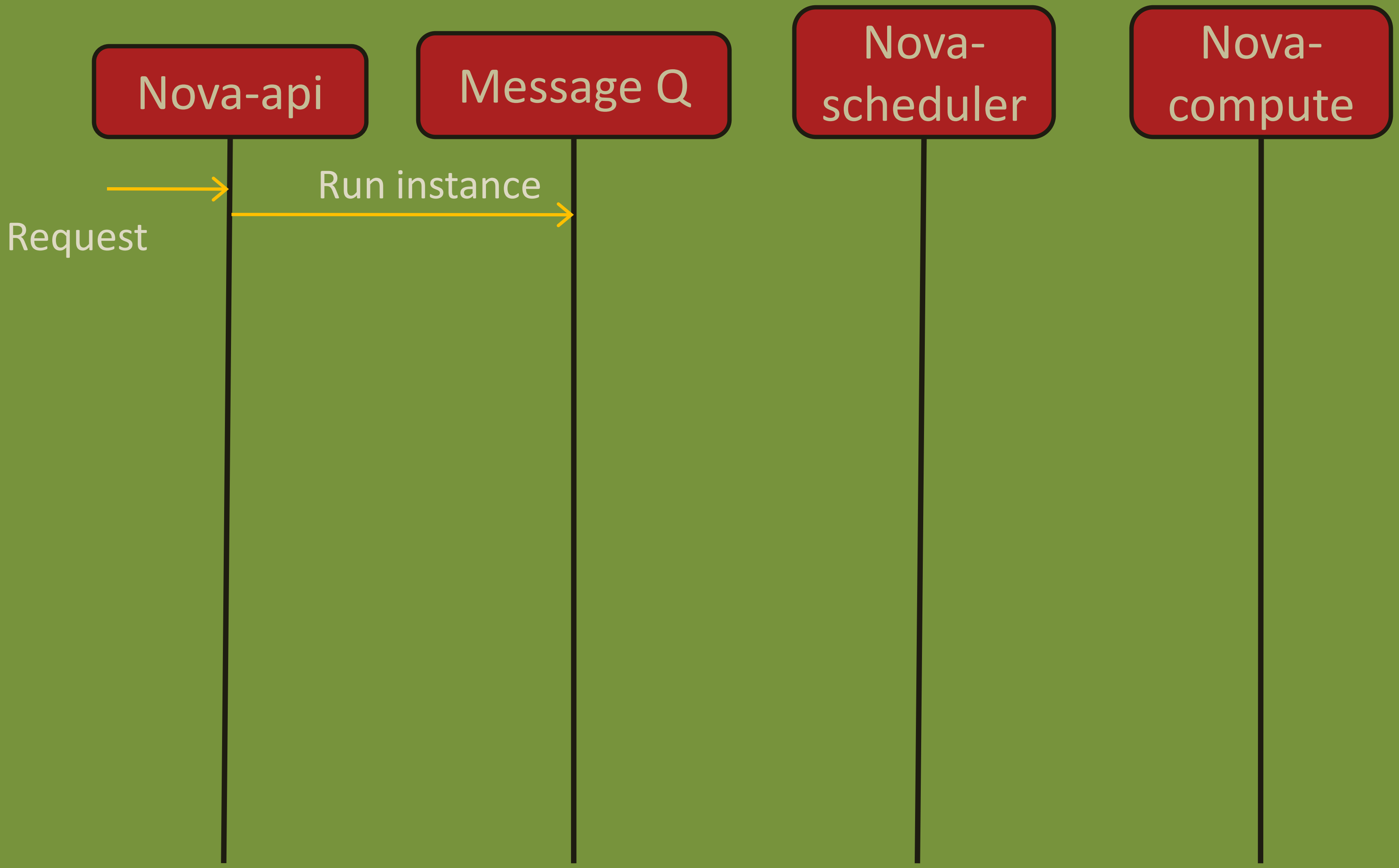
Nova

- Nova handles instances provisioning on compute resources.
- Nova-api initiates most activities
- Nova components communicate via queue and nova database
- Nova-scheduler decides where to launch instances
- Nova-compute launches instances
- Nova-compute periodically report host and network capabilities to Nova-scheduler

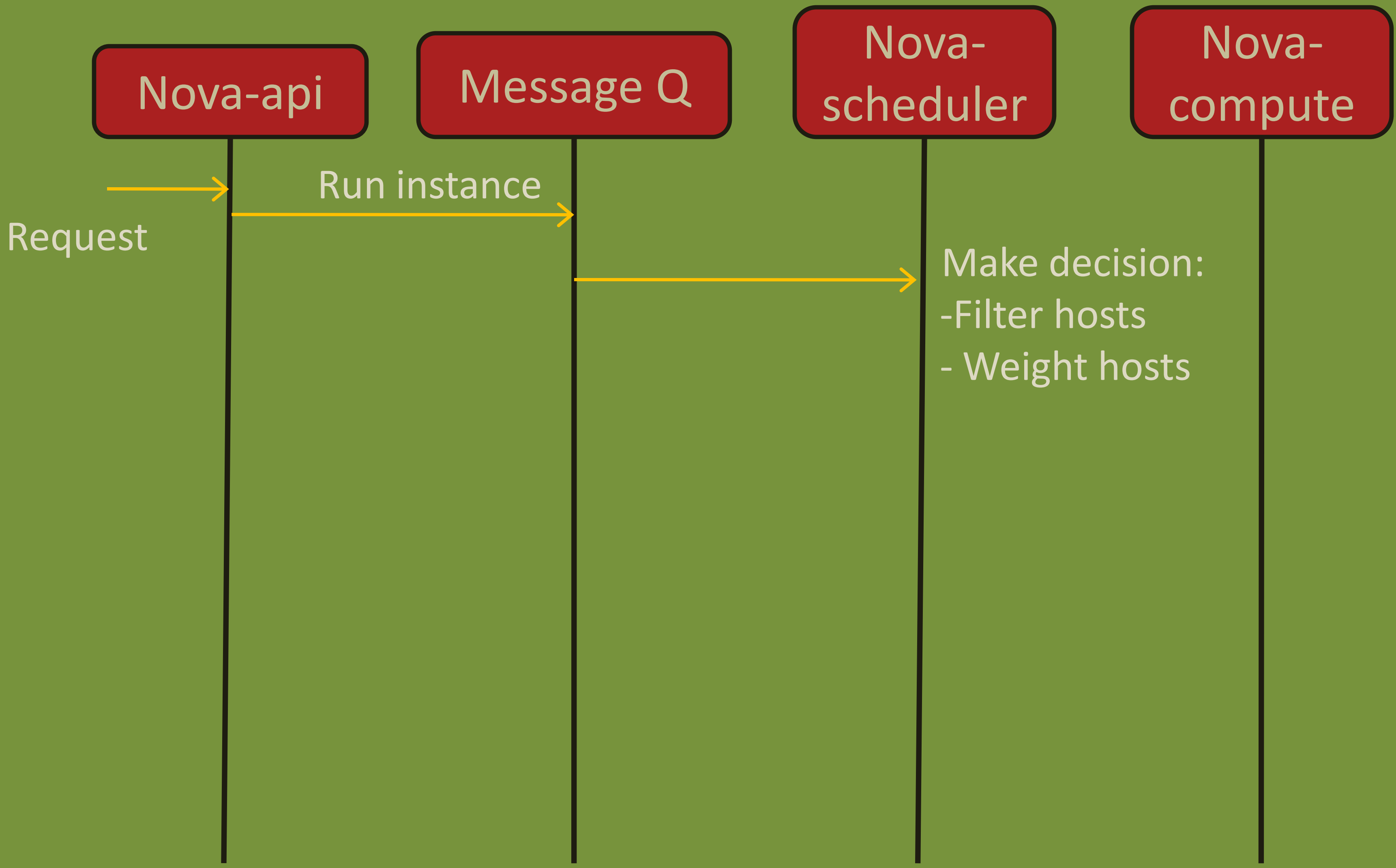
Component Layout



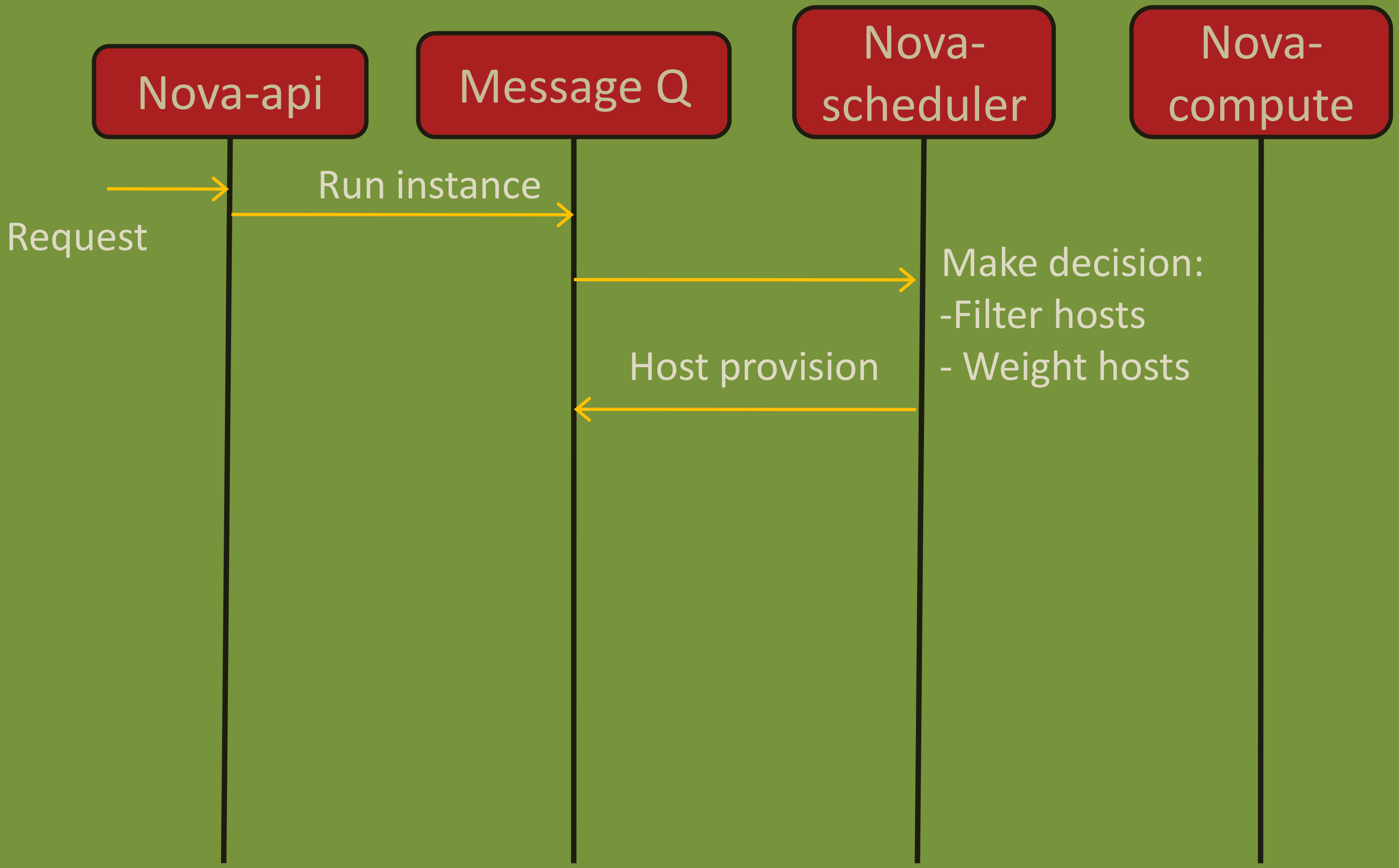
Nova Control Flow



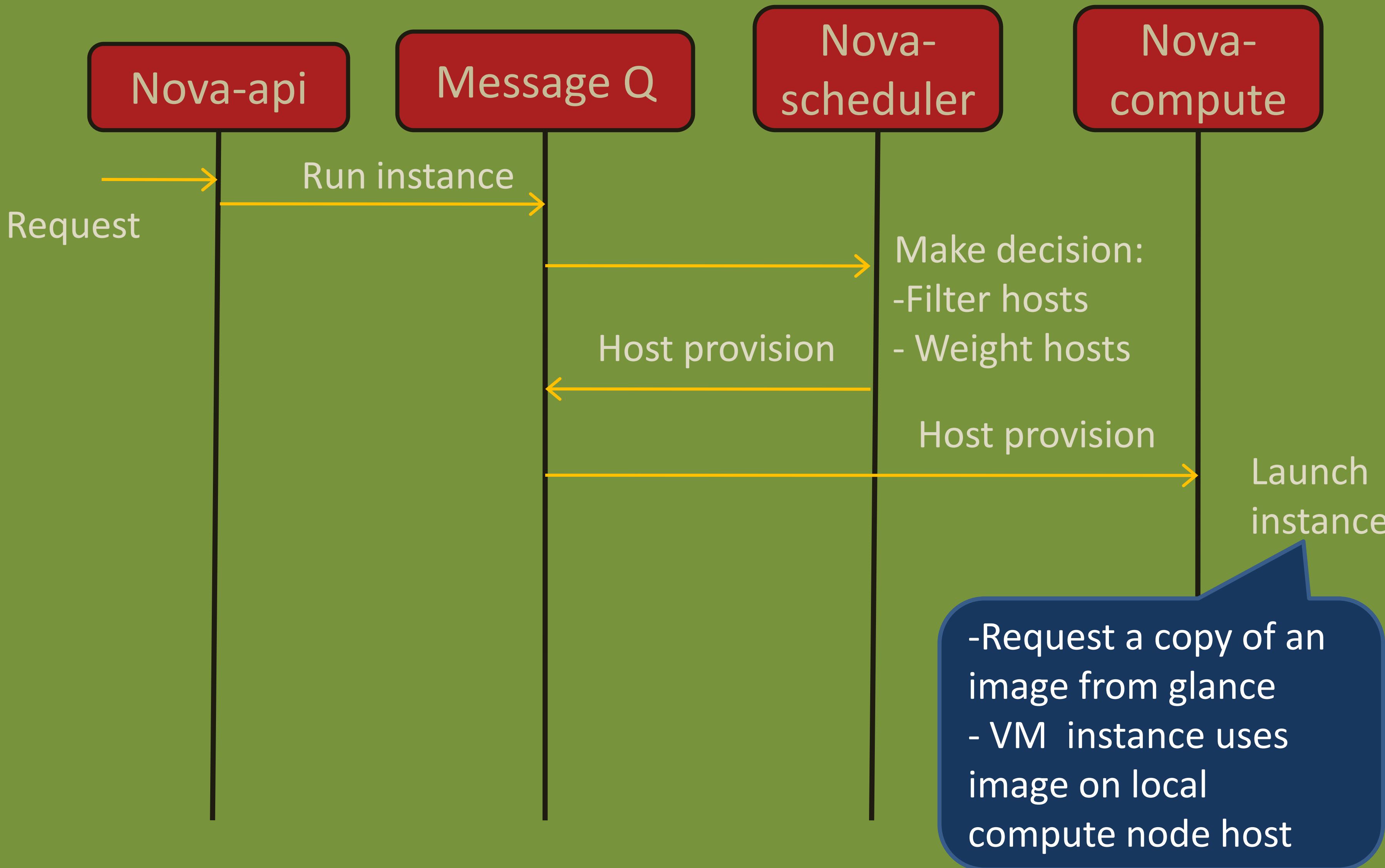
Nova Control Flow



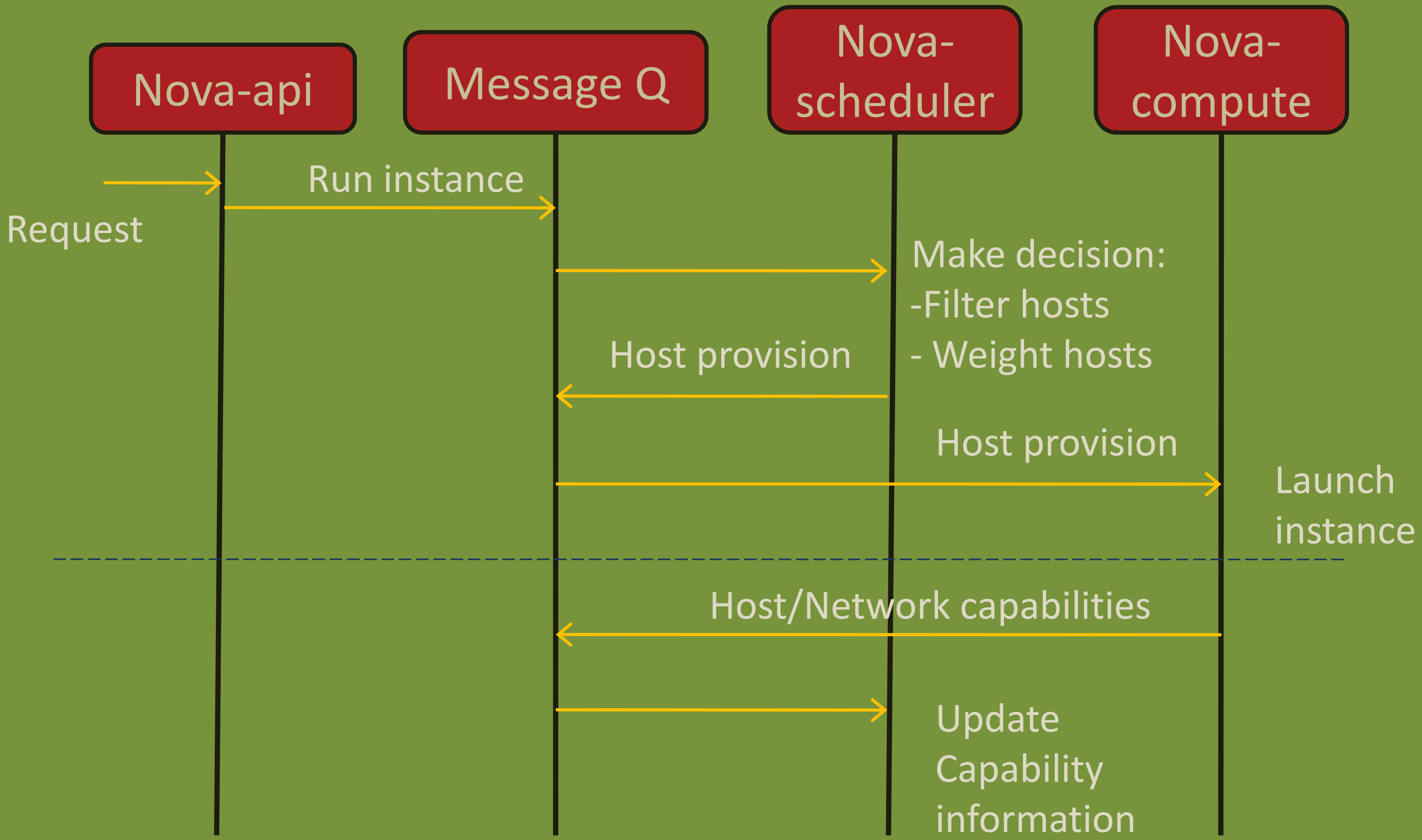
Nova Control Flow

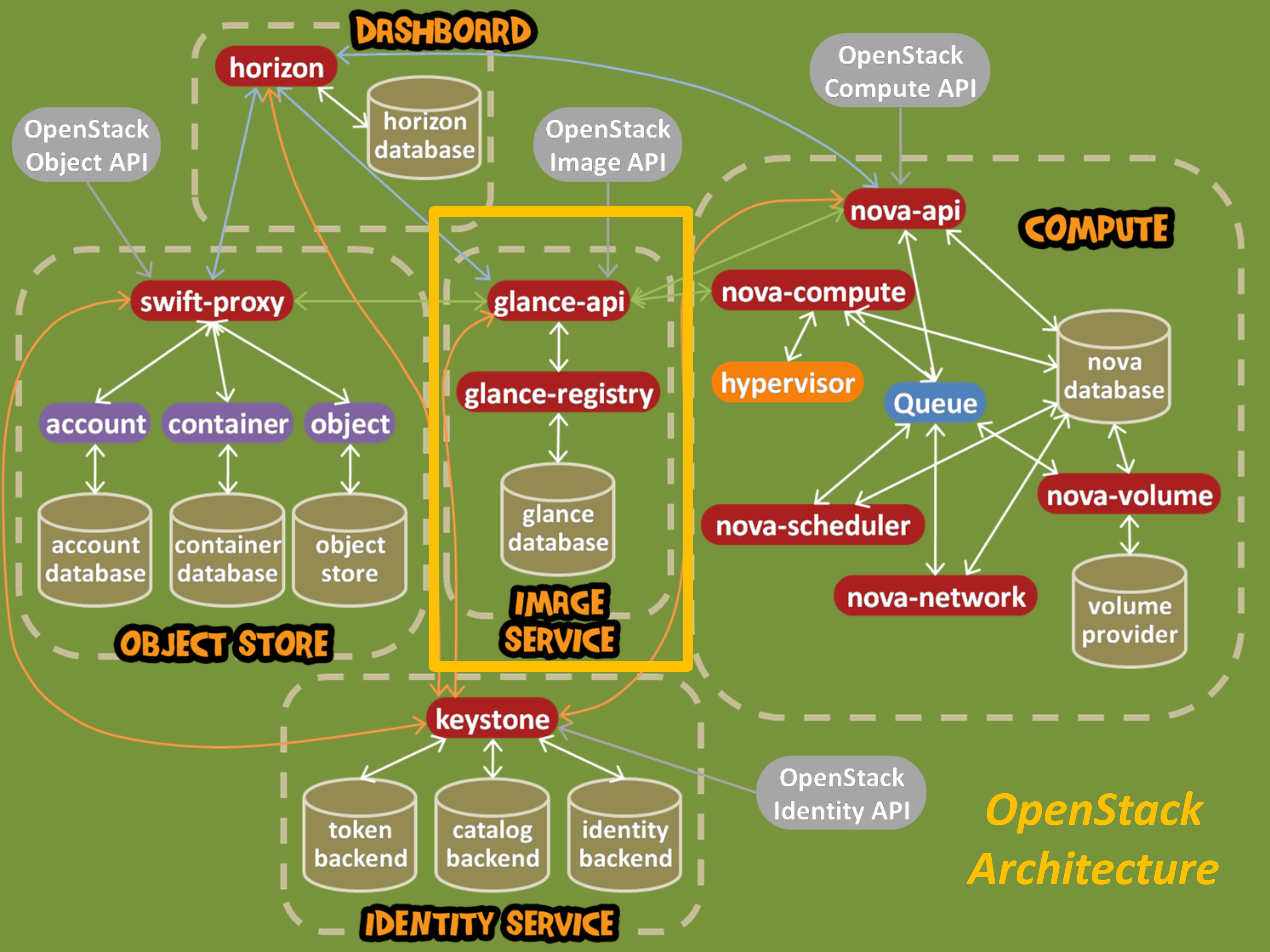


Nova Control Flow



Nova Control Flow



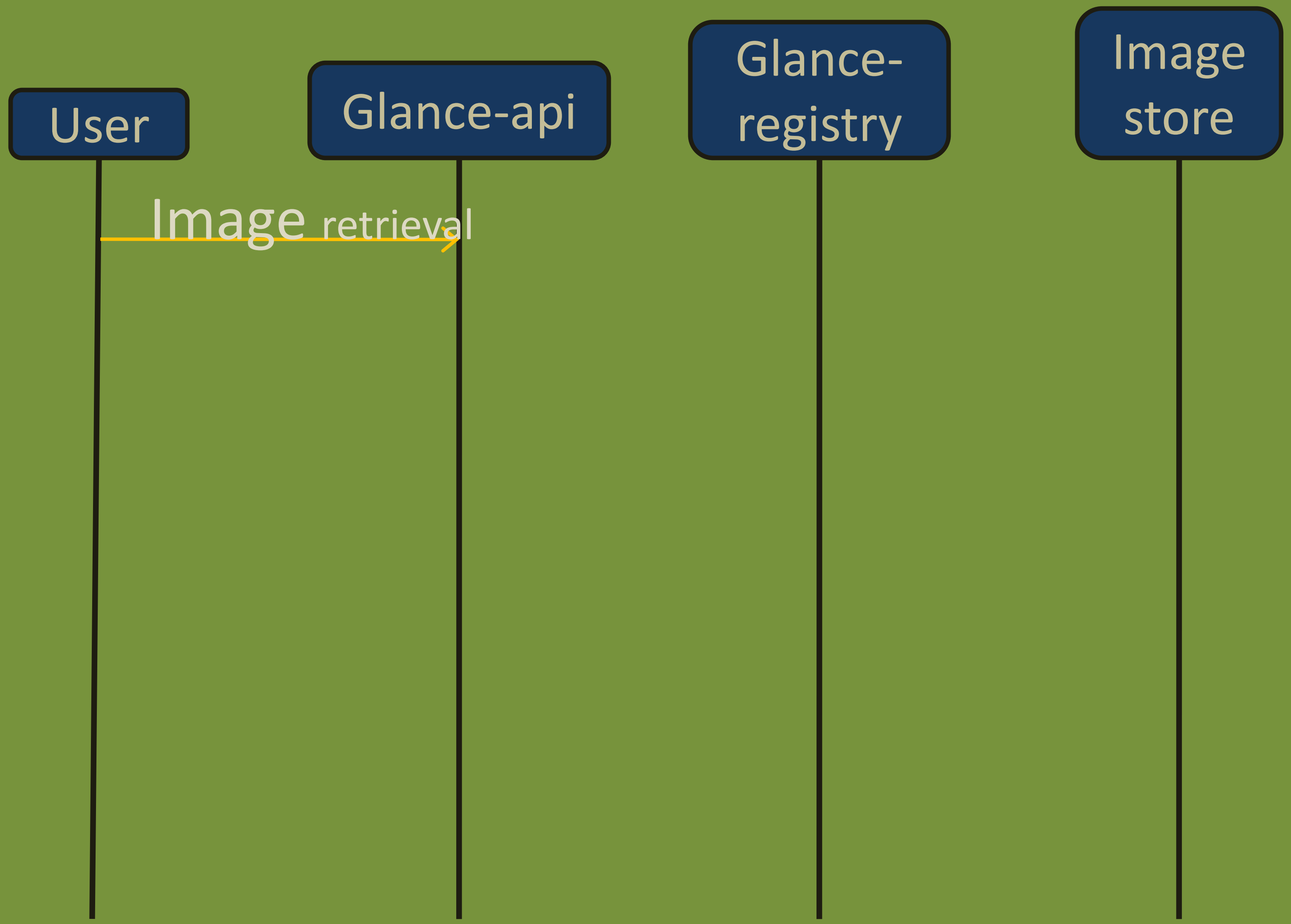


OpenStack Architecture

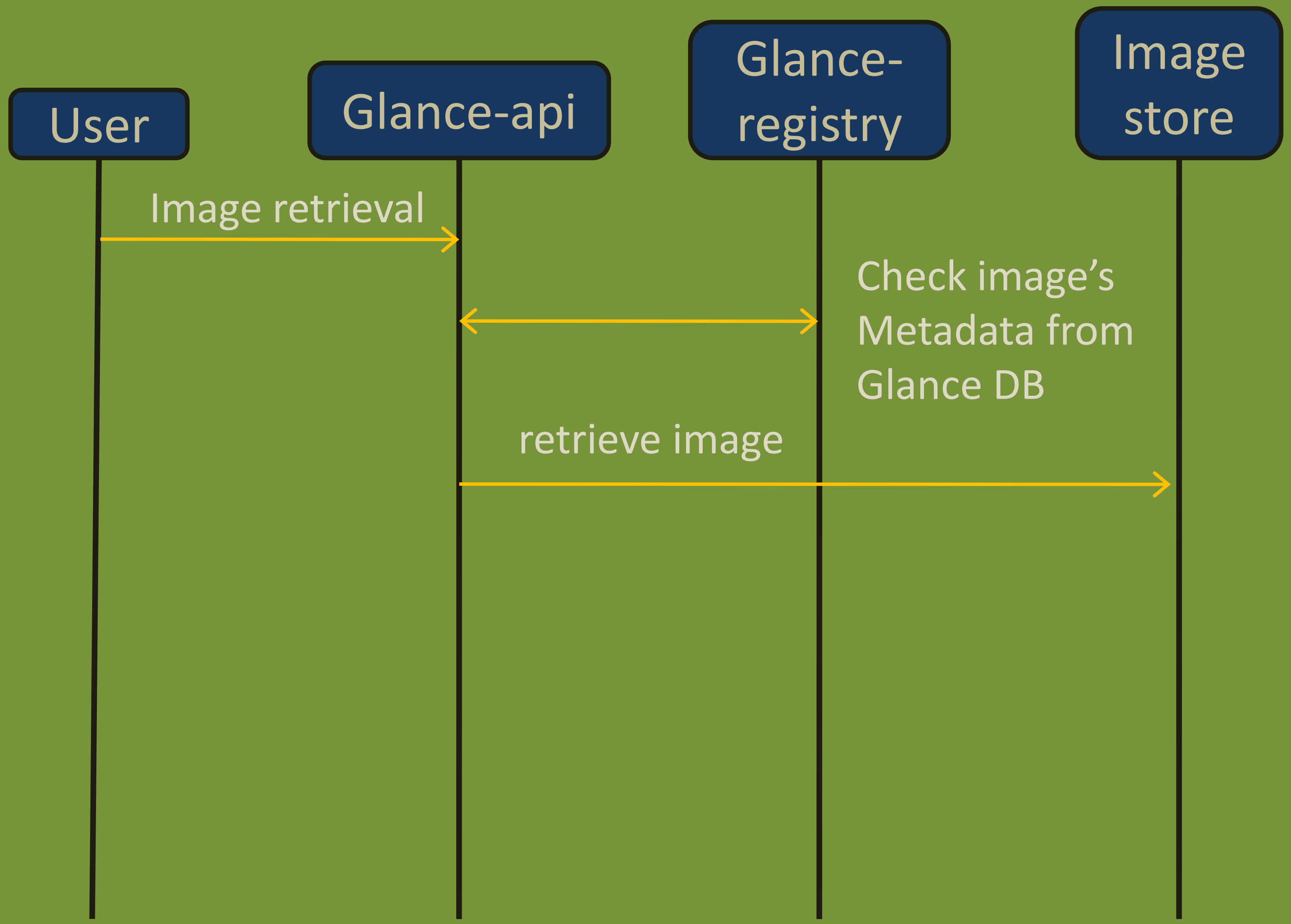
Glance

- Glance manage all kinds of images to instantiate VM instances
- Glance-api takes image retrieval requests from nova-compute and pass them to glance-registry
  - OpenStack create **a new copy** of the image on a host where the VM instance runs
- Glance-registry check image metadata from database
- Glance stores Image data in its image store (S3, HTTP, Local, Swift)

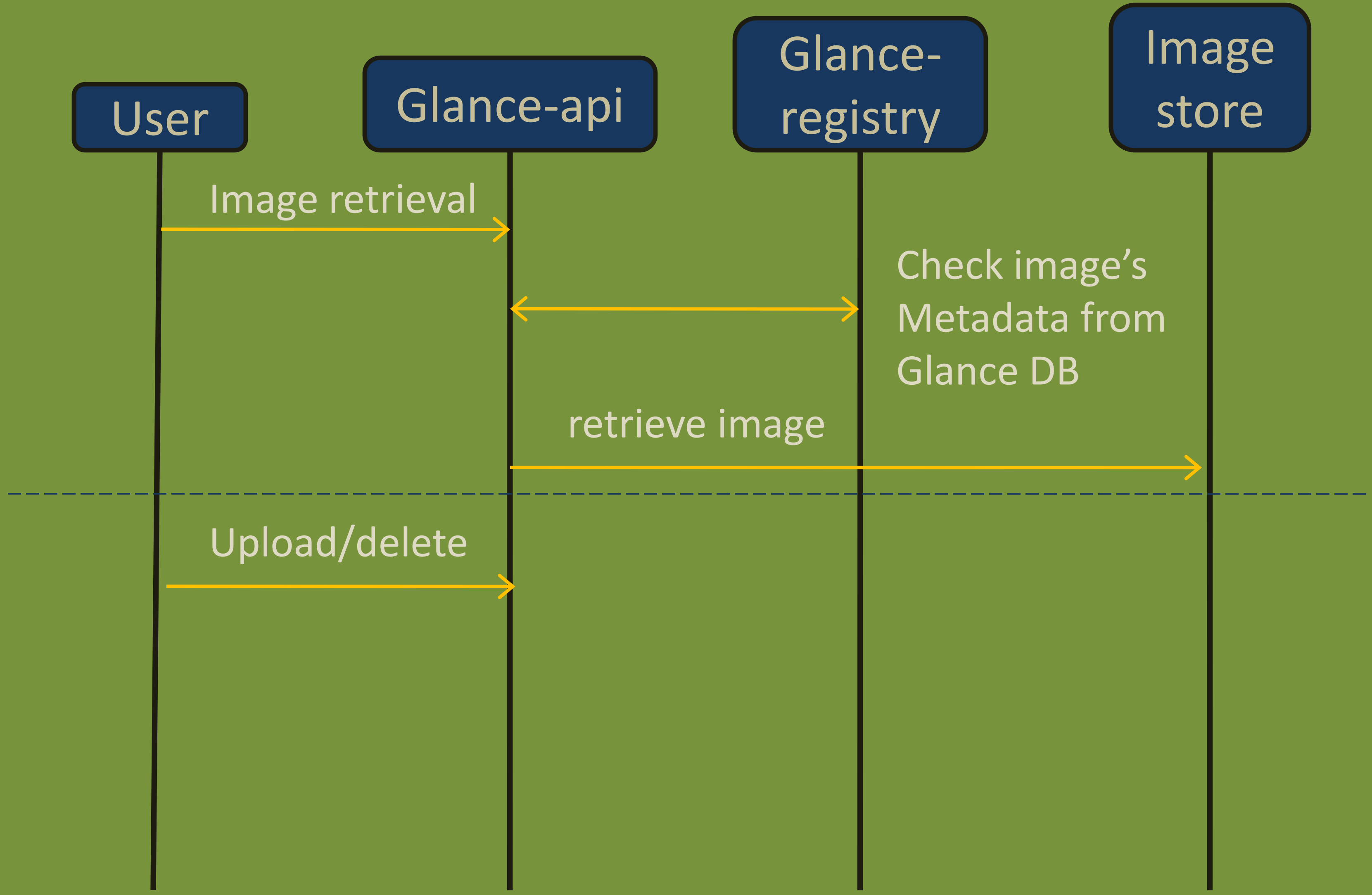
Glance Control Flow



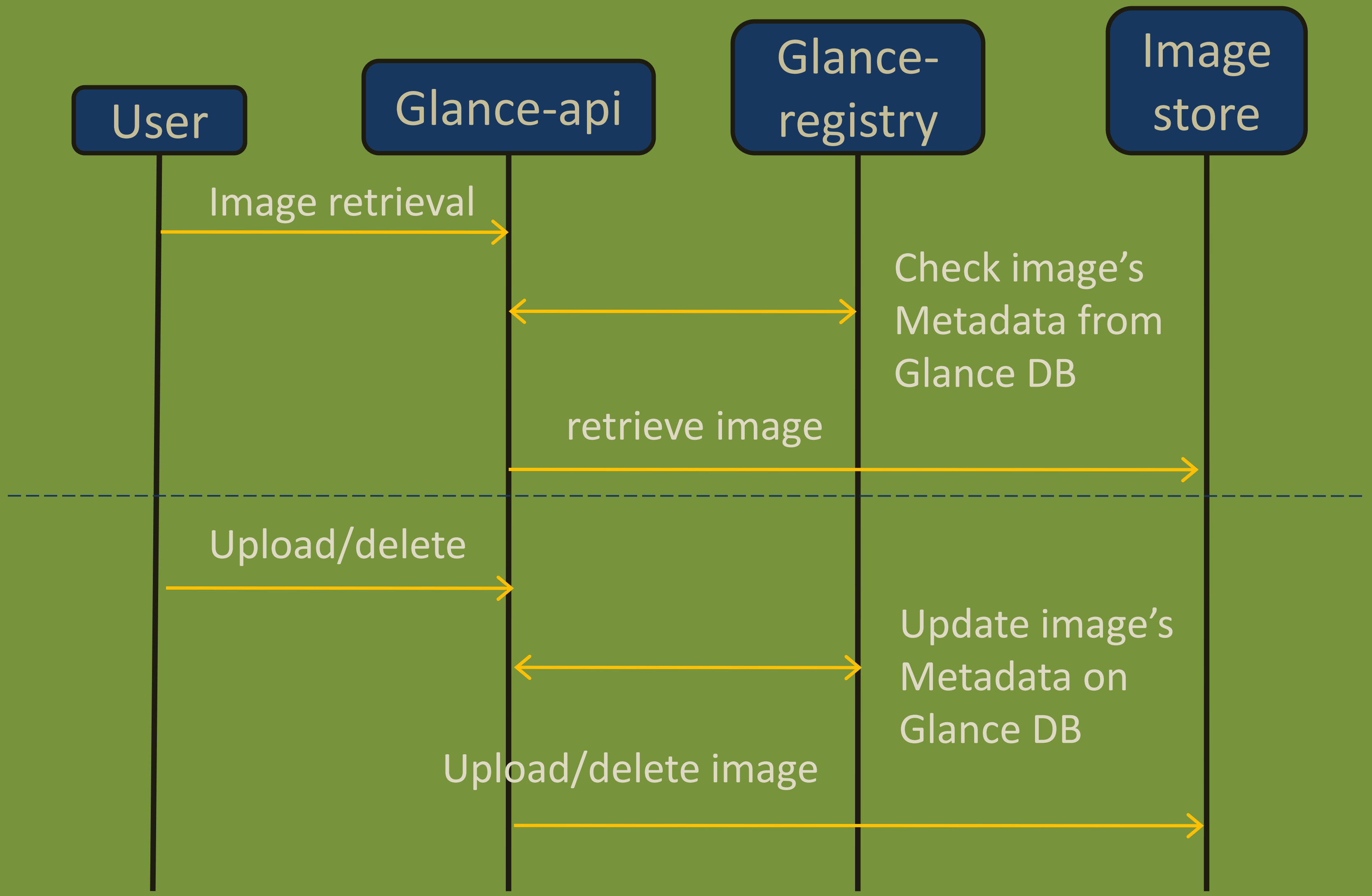
Glance Control Flow



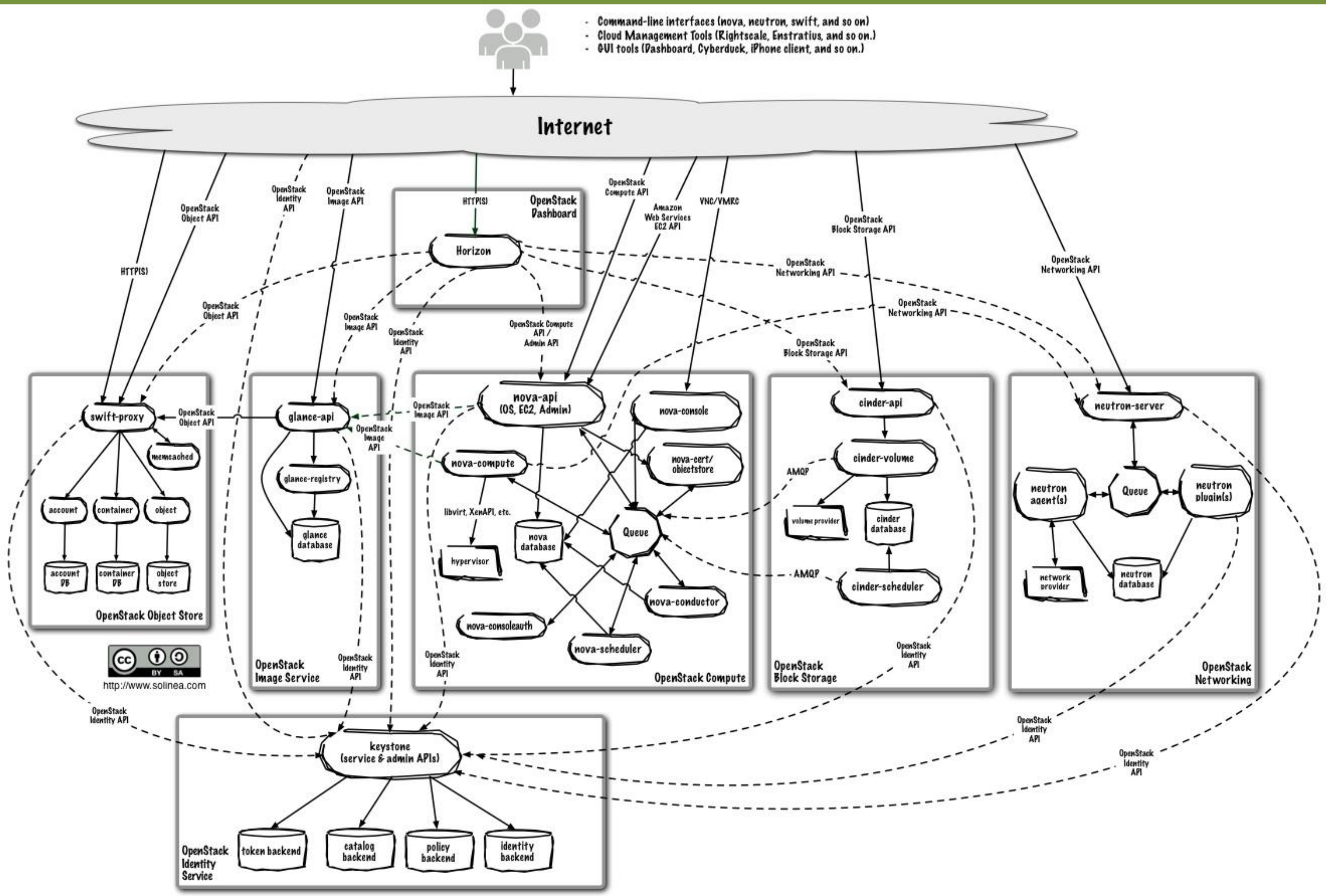
Glance Control Flow

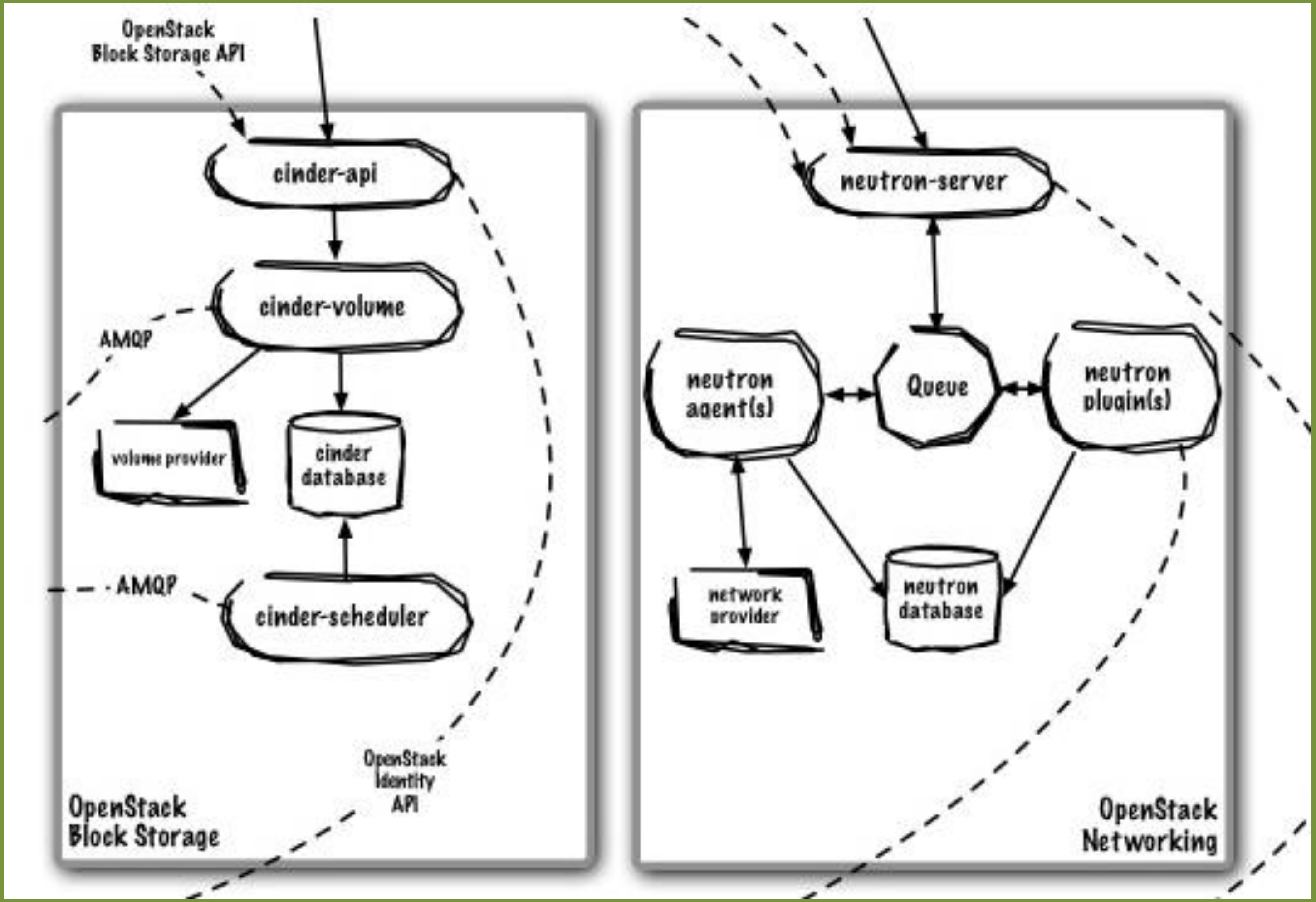


Glance Control Flow



OpenStack Architecture

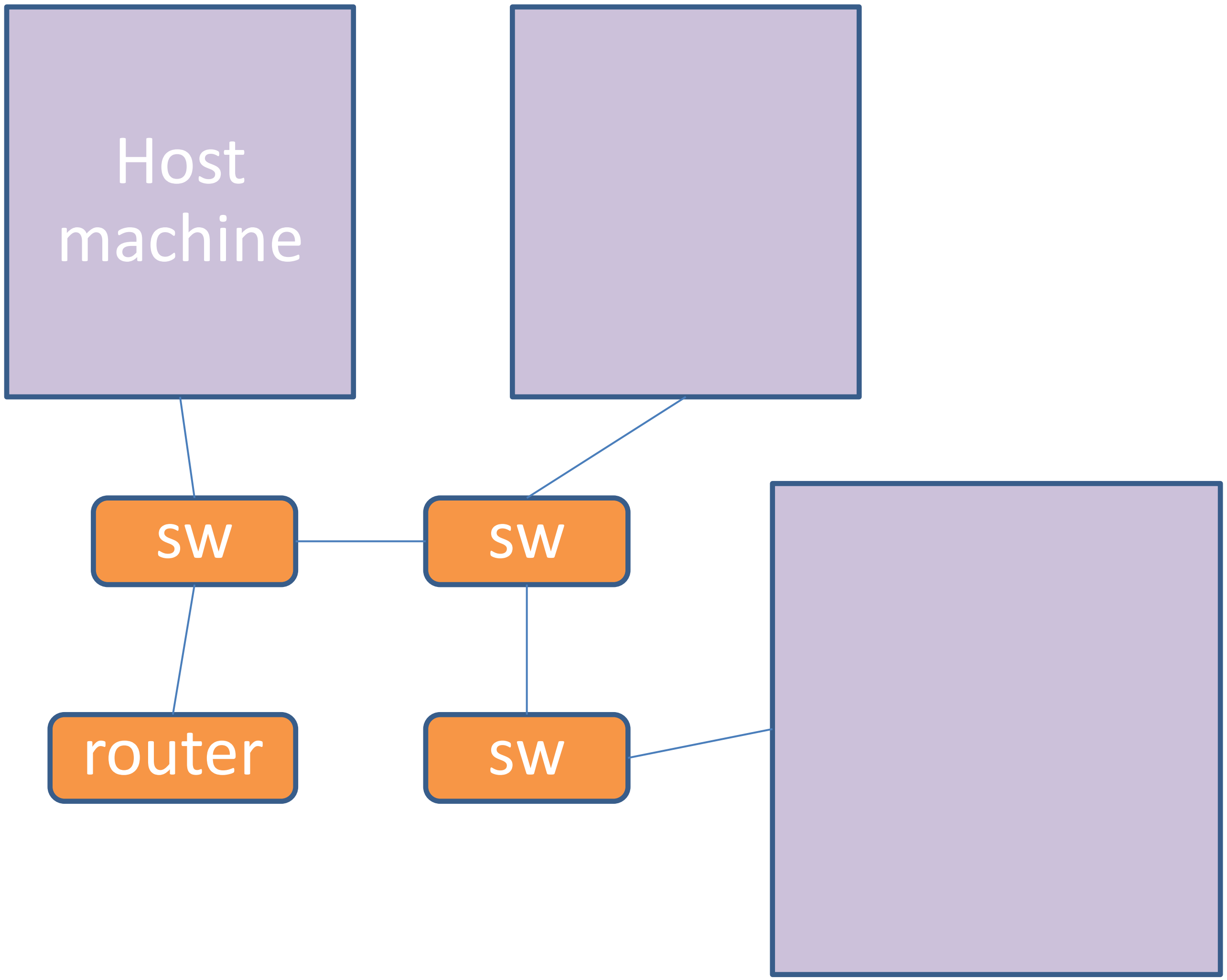



OpenStack
Block Storage API
cinder-api
cinder-volume
AMQP
volume provider
cinder
database
AMQP
cinder-scheduler
OpenStack
Identity
API
OpenStack
Block Storage
neutron-server
neutron
agent(s)
Queue
neutron
plugin(s)
network
provider
neutron
database
OpenStack
Networking

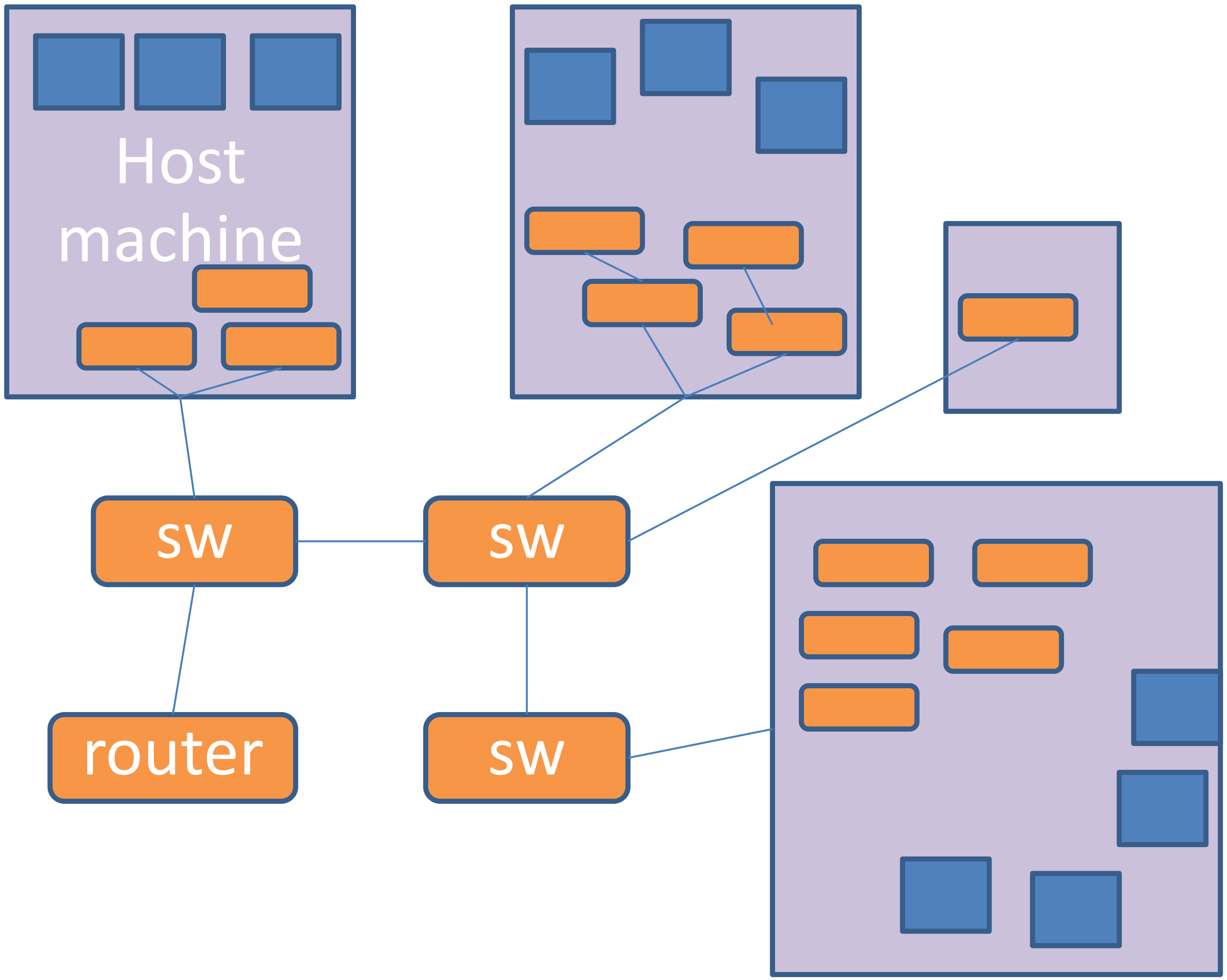
Neutron

- Neutron handle network virtualization
- It allows user/tenant to create logical networks to fit the application's need.
- Use REST API to manage network functions
- Multi-tenancy: Isolation, Abstarction
- Modular: Support Plug-in from multiple vendors

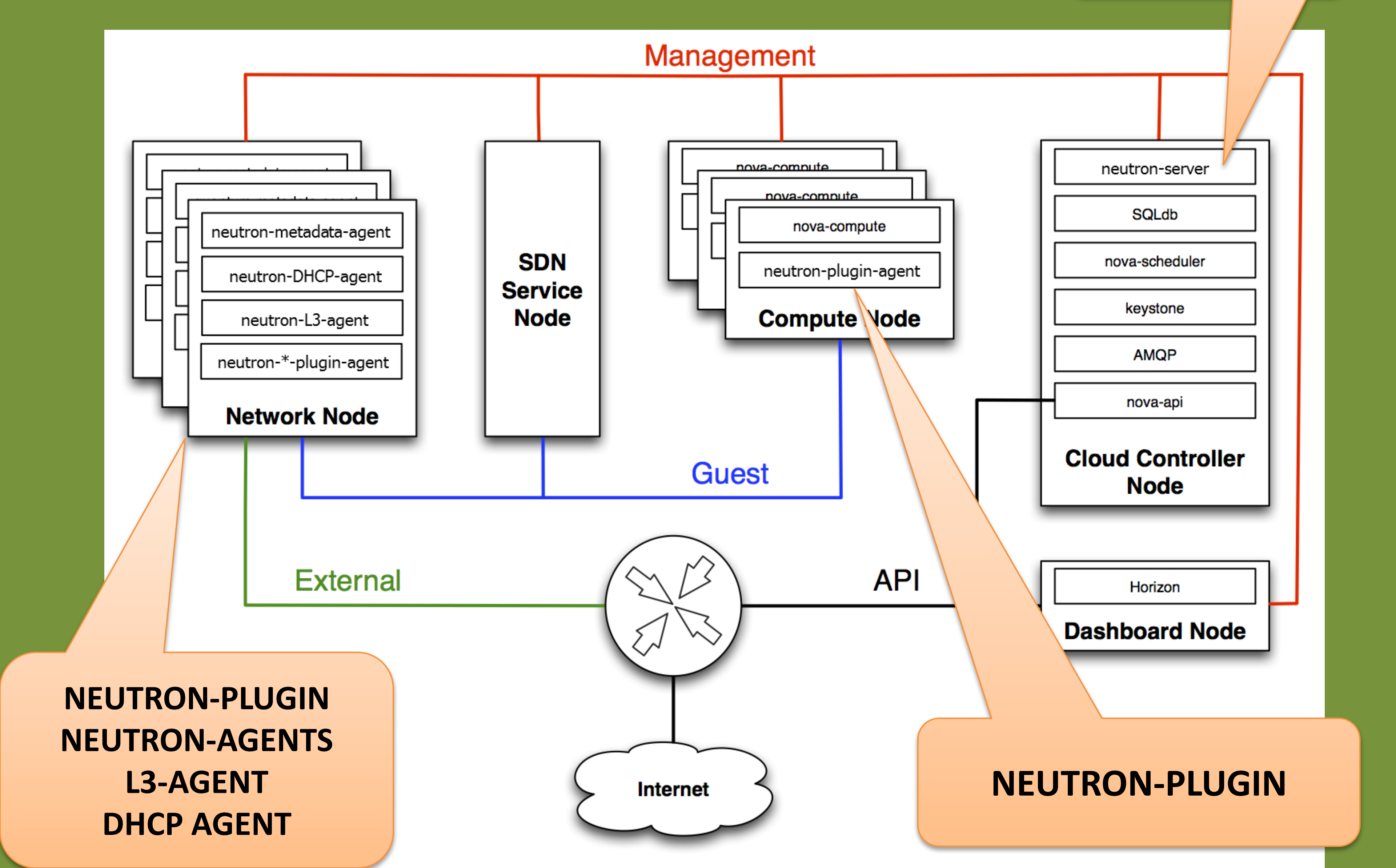
Traditional Host & Network



Moving Networking toward the Edge



Component Layout

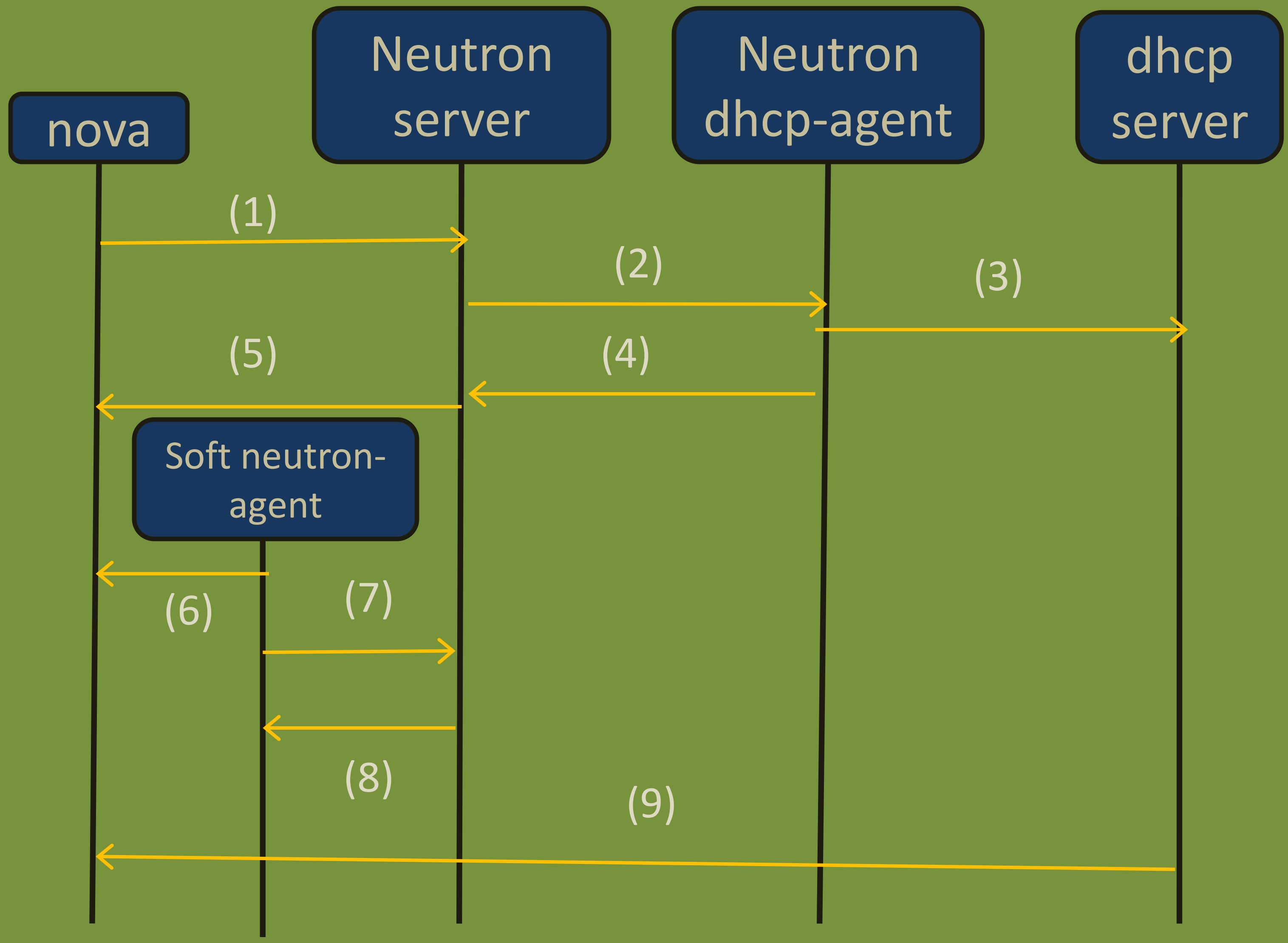


VM booting workflow between nova and neutron

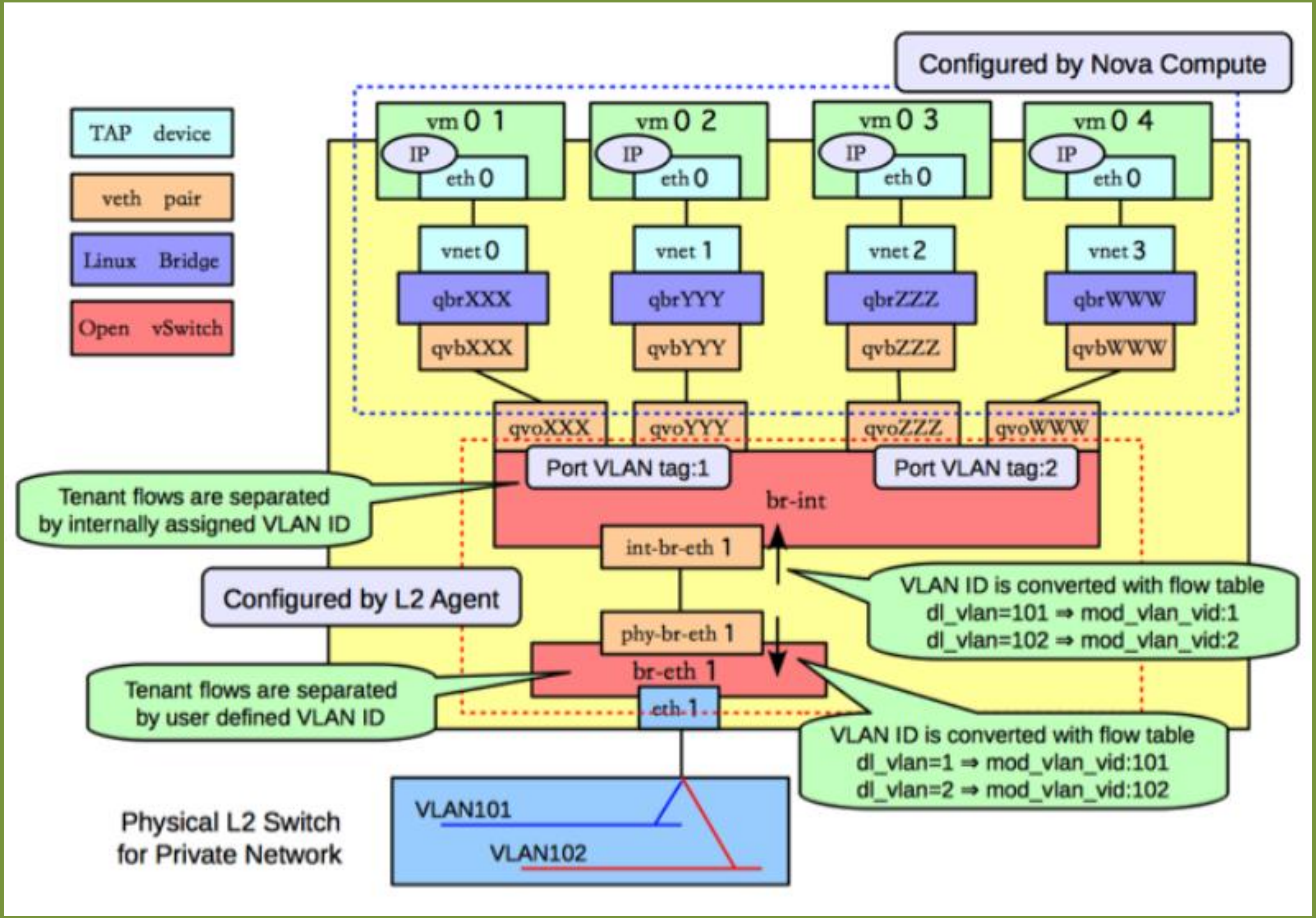
1. **nova** boot will get into compute driver, which will call neutron api to create port
2. **neutron-server** creates the port object and allocates it with ip address from subnets
3. **neutron-server** notifies neutron-dhcp agent with the created port object
4. **neutron-dhcp agent** configs the dhcp server with the port object, such as IP, Mac, gateway and routes
5. **compute-driver gets** the network information, and then create port on br-int soft-switch, and then starts the VM with a tap device attached on the soft-switch port.
6. **soft-neutron-agent** detects and gets to know there is a new soft-switch port created
7. **soft-neutron-agent** asks information from neutron-server
8. **soft-neutron-agent** set up the port, such as the flows and vlan id of the soft-switch port. After this step, the VM's network is connected.
9. **VM** gets the IP address with the dhcp client.

PLUMgrid

Neutron Control Flow

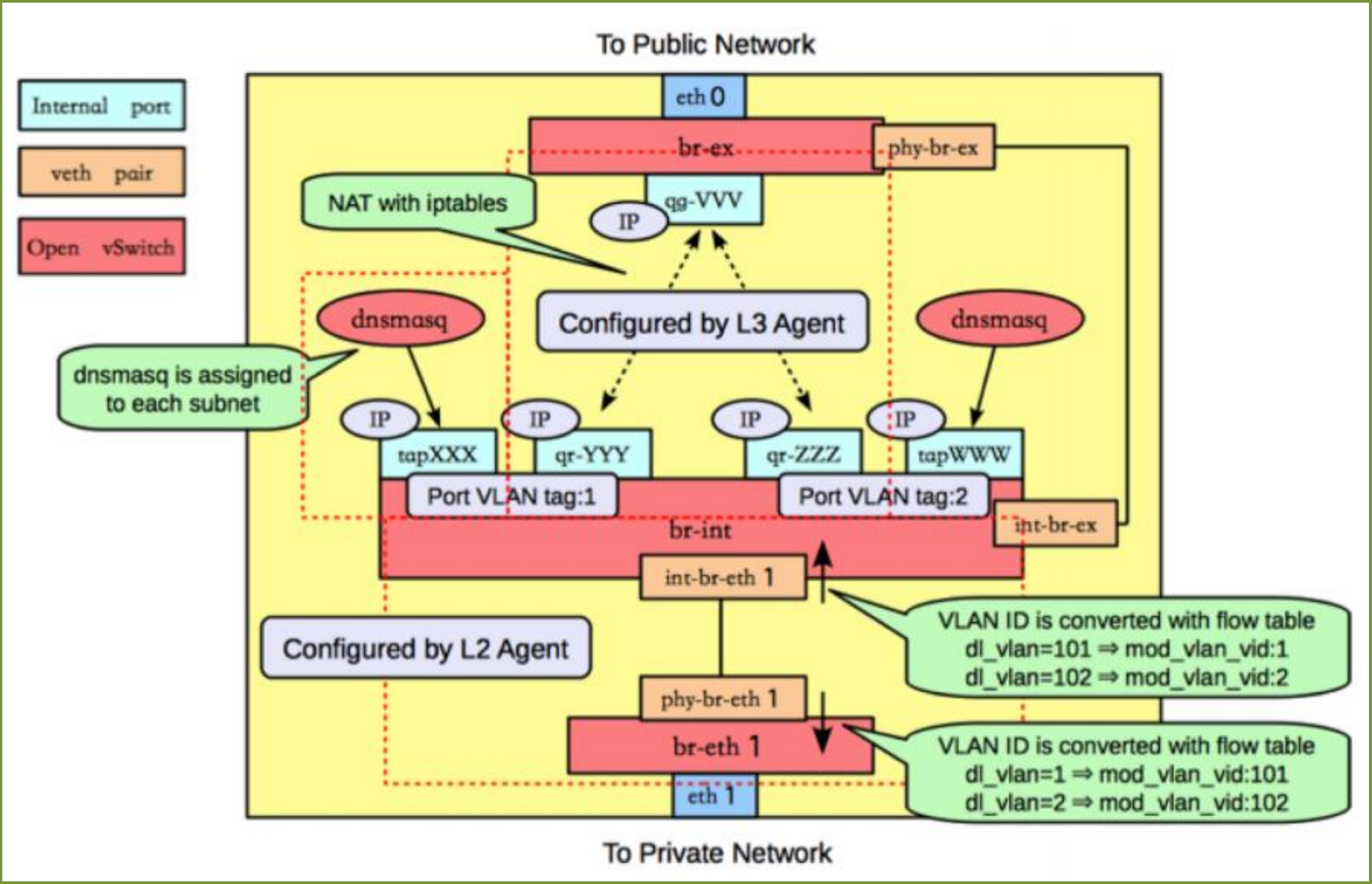


Data flow



Configured by Nova Compute
TAP device
veth pair
Linux Bridge
Open vSwitch
vm01
vm02
vm03
vm04
IP
eth0
vnet0
vnet1
vnet2
vnet3
qbrXXX
qbrYYY
qbrZZZ
qbrWWW
qvbXXX
qvbYYY
qvbZZZ
qvbWWW
qvoXXX
qvoYYY
qvoZZZ
qvoWWW
Port VLAN tag:1
Port VLAN tag:2
br-int
Tenant flows are separated by internally assigned VLAN ID
int-br-eth1
Configured by L2 Agent
VLAN ID is converted with flow table
dl_vlan=101 ⇒ mod_vlan_vid:1
dl_vlan=102 ⇒ mod_vlan_vid:2
phy-br-eth1
br-eth1
Tenant flows are separated by user defined VLAN ID
eth1
VLAN ID is converted with flow table
dl_vlan=1 ⇒ mod_vlan_vid:101
dl_vlan=2 ⇒ mod_vlan_vid:102
VLAN101
VLAN102
Physical L2 Switch for Private Network

Data flow


To Public Network
Internal port
veth pair
Open vSwitch
eth 0
br-ex
phy-br-ex
NAT with iptables
qg-VVV
IP
Configured by L3 Agent
dnsmasq
dnsmasq
dnsmasq is assigned
to each subnet
IP
IP
IP
IP
tapXXX
qr-YYY
qr-ZZZ
tapWWW
Port VLAN tag:1
Port VLAN tag:2
br-int
int-br-ex
int-br-eth 1
VLAN ID is converted with flow table
dl_vlan=101 ⇒ mod_vlan_vid:1
dl_vlan=102 ⇒ mod_vlan_vid:2
Configured by L2 Agent
phy-br-eth 1
br-eth 1
VLAN ID is converted with flow table
dl_vlan=1 ⇒ mod_vlan_vid:101
dl_vlan=2 ⇒ mod_vlan_vid:102
eth 1
To Private Network

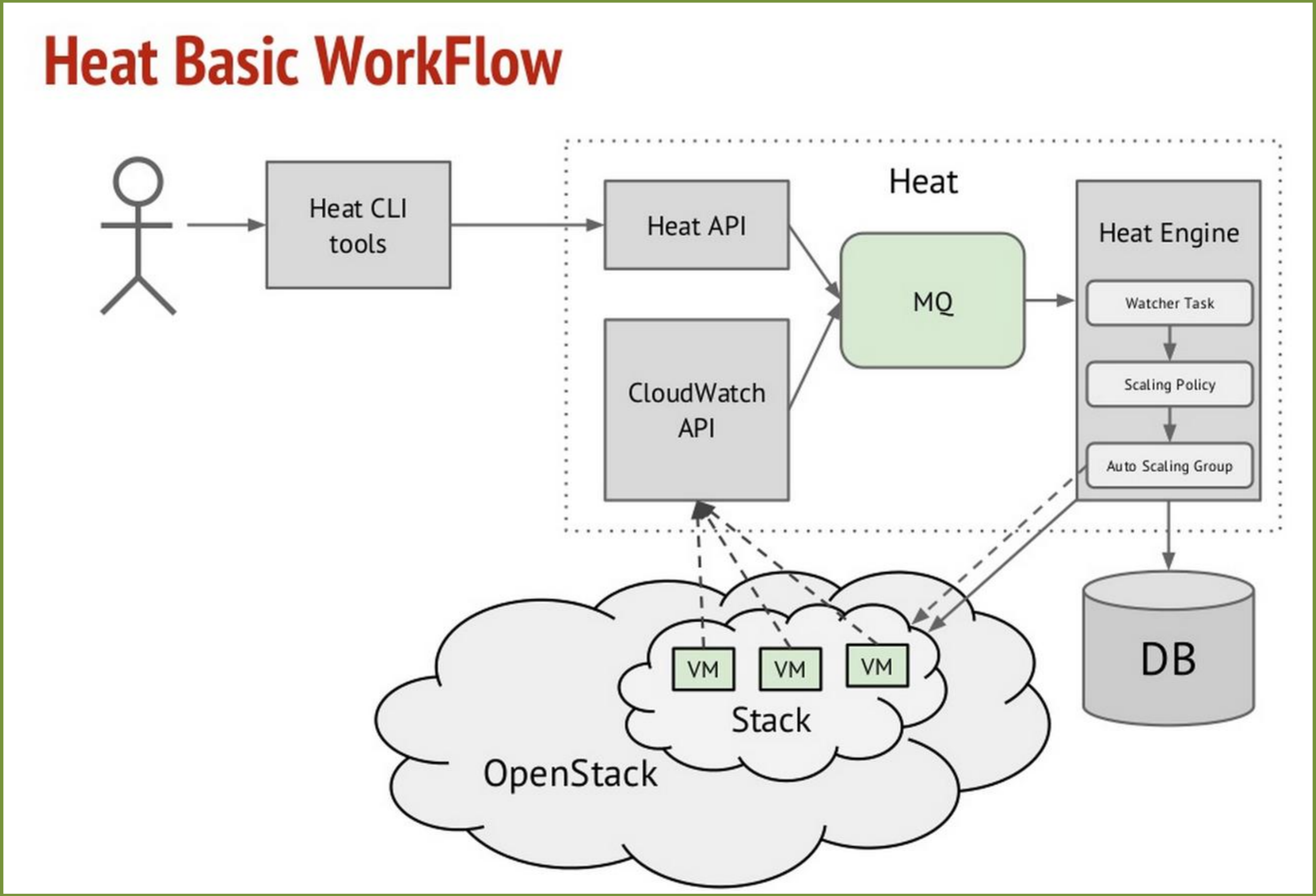
Heat

What is Heat?

- Orchestration service for OpenStack (officially supports Grizzly release)
- Uses templating mechanism
- Controls complex groups of cloud resources
- Huge potential and multiple use cases
- More than 20 active contributors

http://www.slideshare.net/mirantis/an-introduction-to-openstack-heat

Slide taken from mirantis



http://www.slideshare.net/mirantis/an-introduction-to-openstack-heat

Heat Components

Heat API

- heat-api (OpenStack native REST API) or heat-api-cfn (provides AWS Query API)
- Communicates with Heat Engine and tells it what actions to do

Heat Engine

- Does all the orchestration work
- Layer on which resource integration is implemented
- Contains abstractions to use Auto Scaling and High Availability

http://www.slideshare.net/mirantis/an-introduction-to-openstack-heat

HOT format

- *Stack:* In Heat parlance, a stack is the collection of objects—or *resources*—that will be created by Heat. This might include instances (VMs), networks, subnets, routers, ports, router interfaces, security groups, security group rules, auto-scaling rules, etc.
- *Template:* Heat uses the idea of a template to define a stack. If you wanted to have a stack that created two instances connected by a private network, then your template would contain the definitions for two instances, a network, a subnet, and two network ports. Since templates are central to how Heat operates, I'll show you examples of templates in this post.
- *Parameters:* A Heat template has three major sections, and one of those sections defines the template's parameters. These are tidbits of information—like a specific image ID, or a particular network ID—that are passed to the Heat template by the user. This allows users to create more generic templates that could potentially use different resources.
- *Resources:* Resources are the specific objects that Heat will create and/or modify as part of its operation, and the second of the three major sections in a Heat template.

From www.openstack.org

Basic HOT

```
heat_template_version: 2013-05-23

description: Test Template

parameters:
  ImageID:
    type: string
    description: Image use to boot a server
  NetID:
    type: string
    description: Network ID for the server

resources:
  server1:
    type: OS::Nova::Server
    properties:
      name: "Test server"
      image: { get_param: ImageID }
      flavor: "m1.tiny"
      networks:
      - network: { get_param: NetID }

outputs:
  server1_private_ip:
    description: IP address of the server in the private network
    value: { get_attr: [ server1, first_address ] }
```

From www.openstack.org

```
$ NET_ID=$(nova net-list | awk '/ demo-net / { print $2 }')
$ heat stack-create -f test-stack.yml \
  -P "ImageID=cirros-0.3.2-x86_64;NetID=$NET_ID" testStack
+--------------------------------------+------------+--------------------+---------
| id                                   | stack_name | stack_status       | creation_
+--------------------------------------+------------+--------------------+---------
| 477d96b4-d547-4069-938d-32ee990834af | testStack  | CREATE_IN_PROGRESS | 2014-04-
+--------------------------------------+------------+--------------------+---------
```

Example HOT

```yaml
heat_template_version: 2013-05-23
description: >
  A simple Heat template that spins up multiple instances and a privat
resources:
  heat_network_01:
    type: OS::Neutron::Net
    properties:
      admin_state_up: true
      name: heat-network-01
  heat_subnet_01:
    type: OS::Neutron::Subnet
    properties:
      name: heat-subnet-01
      cidr: 10.10.10.0/24
      dns_nameservers: [172.16.1.11, 172.16.1.6]
      enable_dhcp: true
      gateway_ip: 10.10.10.254
      network_id: { get_resource: heat_network_01 }
```

From http://blog.scottlowe.org/2014/05/02/another-look-at-an-openstack-heat-template/

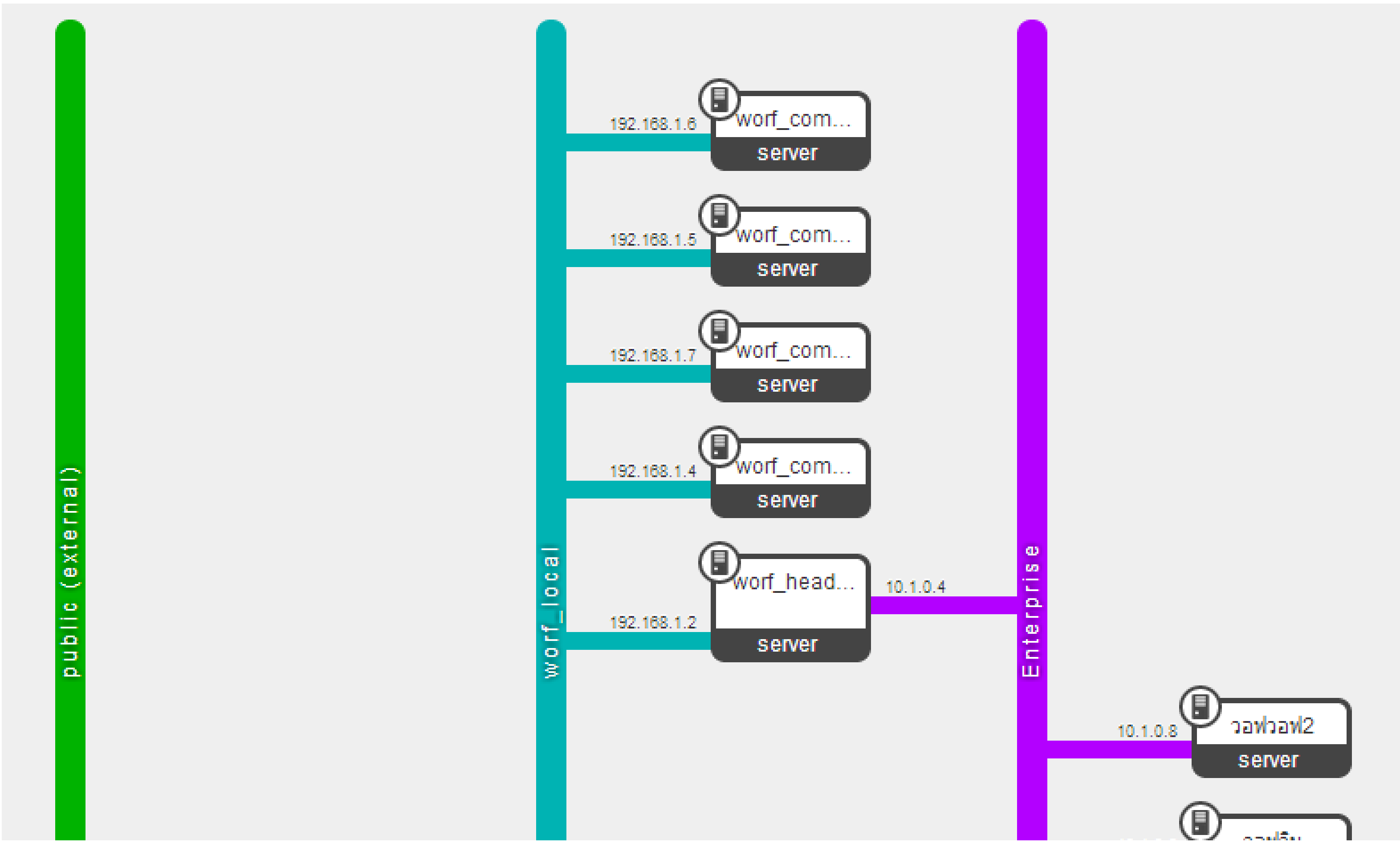
```
  heat_router_01:
    type: OS::Neutron::Router
    properties:
      admin_state_up: true
      name: heat-router-01
  heat_router_01_gw:
    type: OS::Neutron::RouterGateway
    properties:
      network_id: 604146b3-2e0c-4399-826e-a18cbc18362b
      router_id: { get_resource: heat_router_01 }
  heat_router_int0:
    type: OS::Neutron::RouterInterface
    properties:
      router_id: { get_resource: heat_router_01 }
      subnet_id: { get_resource: heat_subnet_01 }
```

```
  instance0_port0:
    type: OS::Neutron::Port
    properties:
      admin_state_up: true
      network_id: { get_resource: heat_network_01 }
      security_groups:
        - b0ab35c3-63f0-48d2-8a6b-08364a026b9c
  instance1_port0:
    type: OS::Neutron::Port
    properties:
      admin_state_up: true
      network_id: { get_resource: heat_network_01 }
      security_groups:
        - b0ab35c3-63f0-48d2-8a6b-08364a026b9c
```

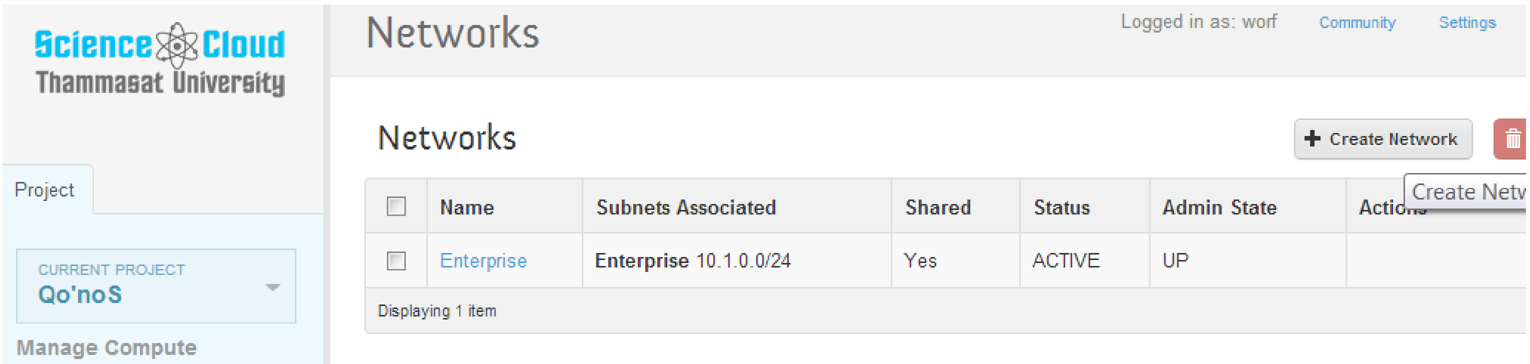
```
  instance0:
    type: OS::Nova::Server
    properties:
      name: heat-instance-01
      image: 01b0eb5d-14ae-4c9e-8025-a21e6f733034
      flavor: m1.xsmall
      networks:
        - port: { get_resource: instance0_port0 }
  instance1:
    type: OS::Nova::Server
    properties:
      name: heat-instance-02
      image: 01b0eb5d-14ae-4c9e-8025-a21e6f733034
      flavor: m1.xsmall
      networks:
        - port: { get_resource: instance1_port0 }
```

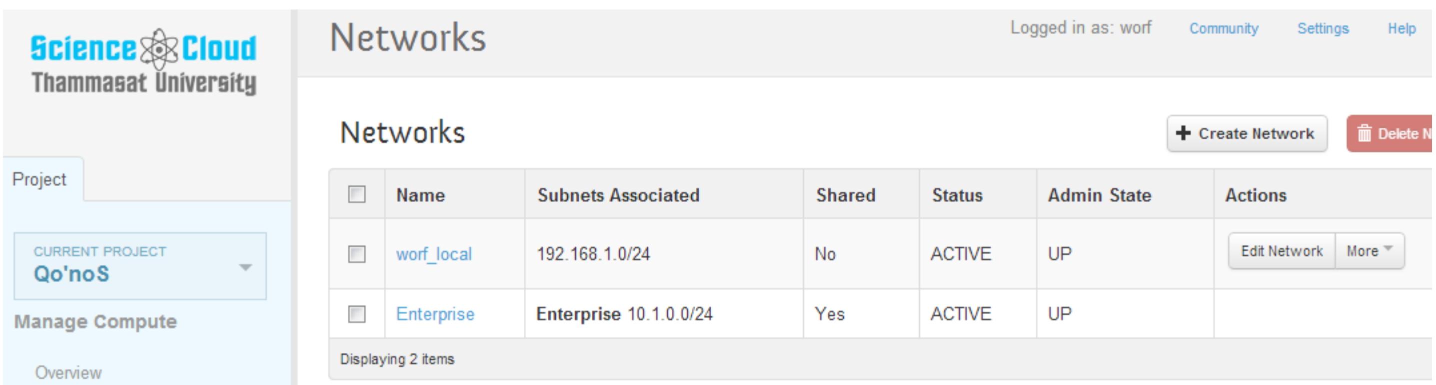
Creating a Cluster Computer on

OpenStack

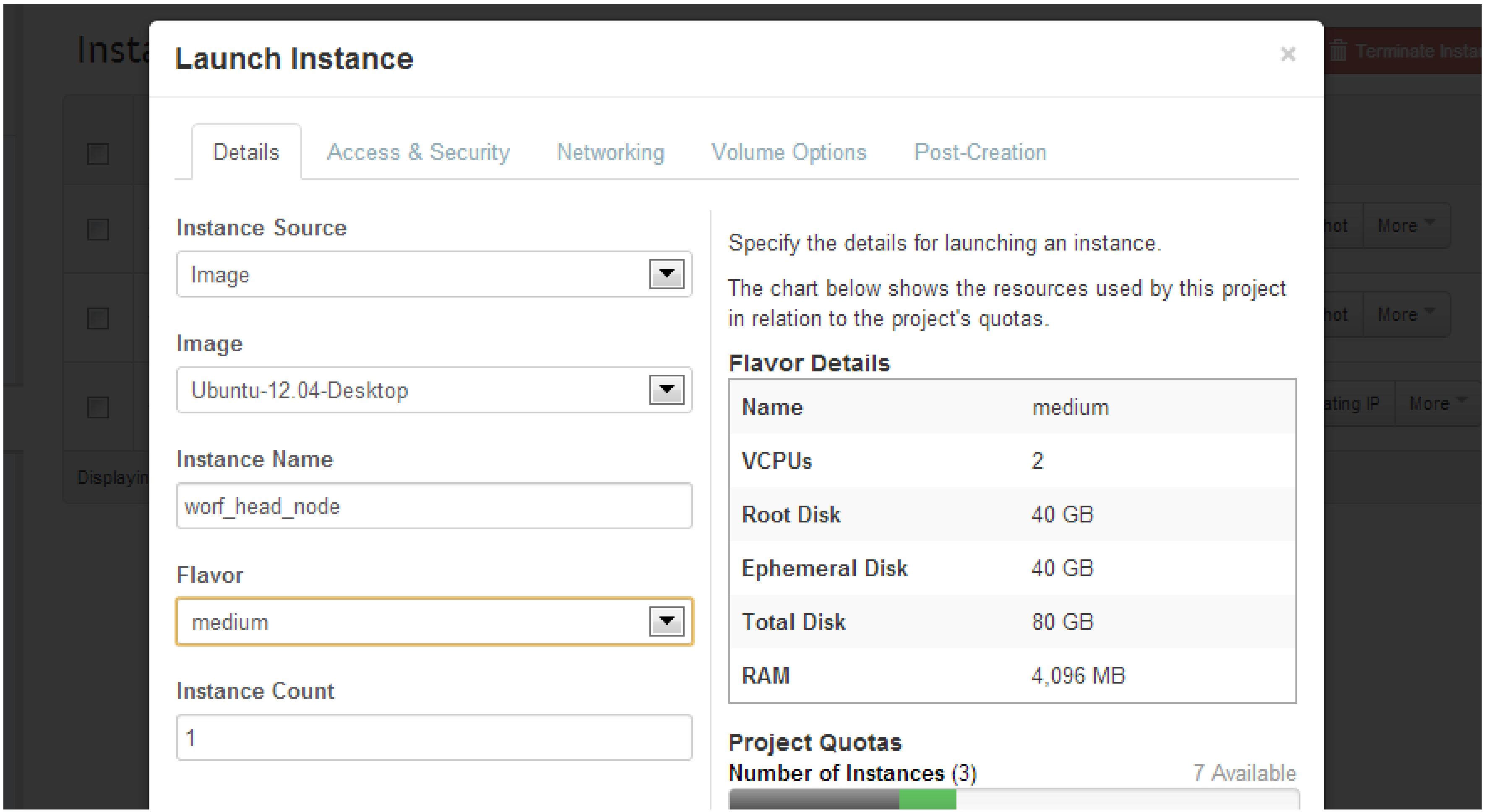

public (external)
worf_local
Enterprise
192.168.1.6
worf_com...
server
192.168.1.5
worf_com...
server
192.168.1.7
worf_com...
server
192.168.1.4
worf_com...
server
192.168.1.2
worf_head...
server
10.1.0.4
10.1.0.8
วอฟวอฟ2
server

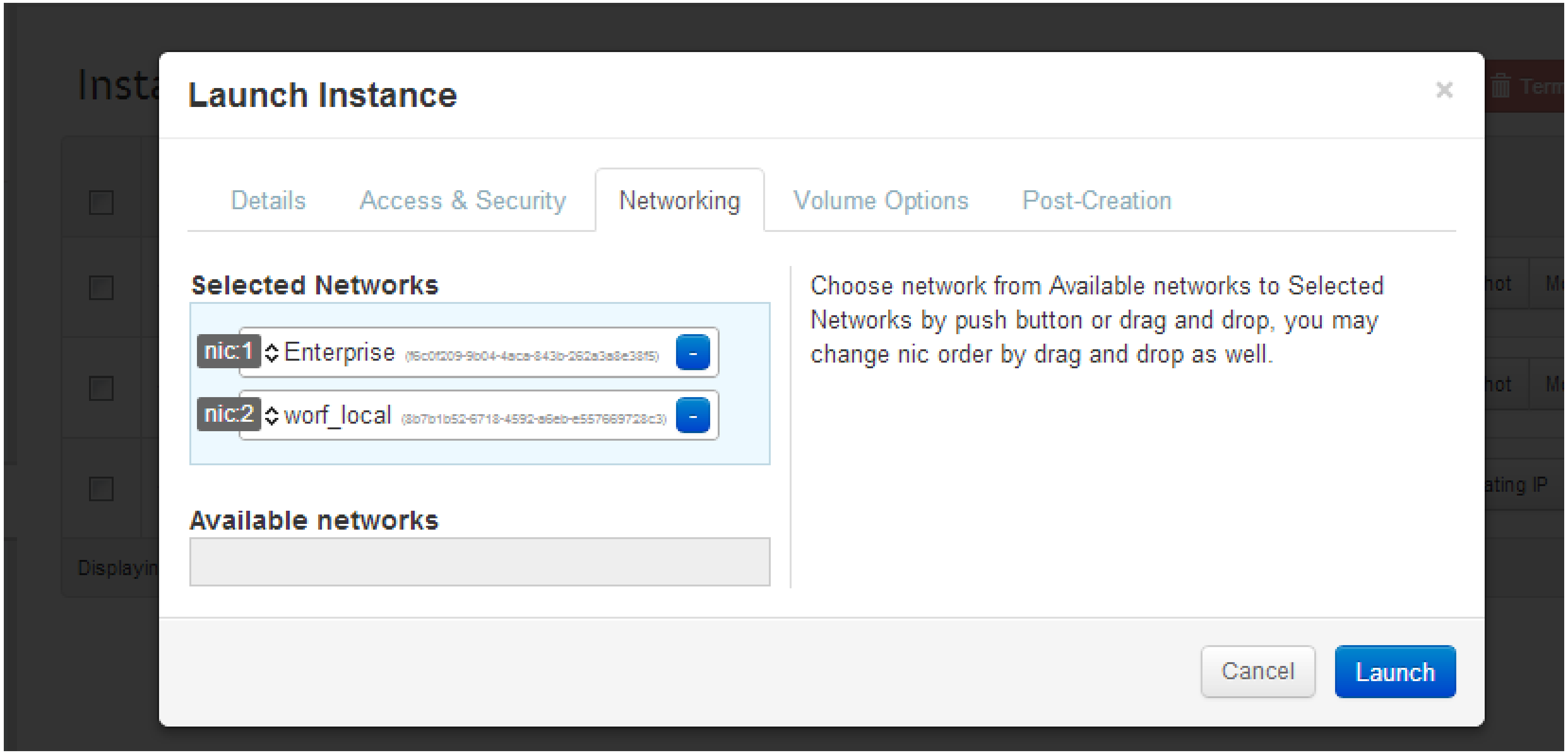
Create a local network





Launch a head node



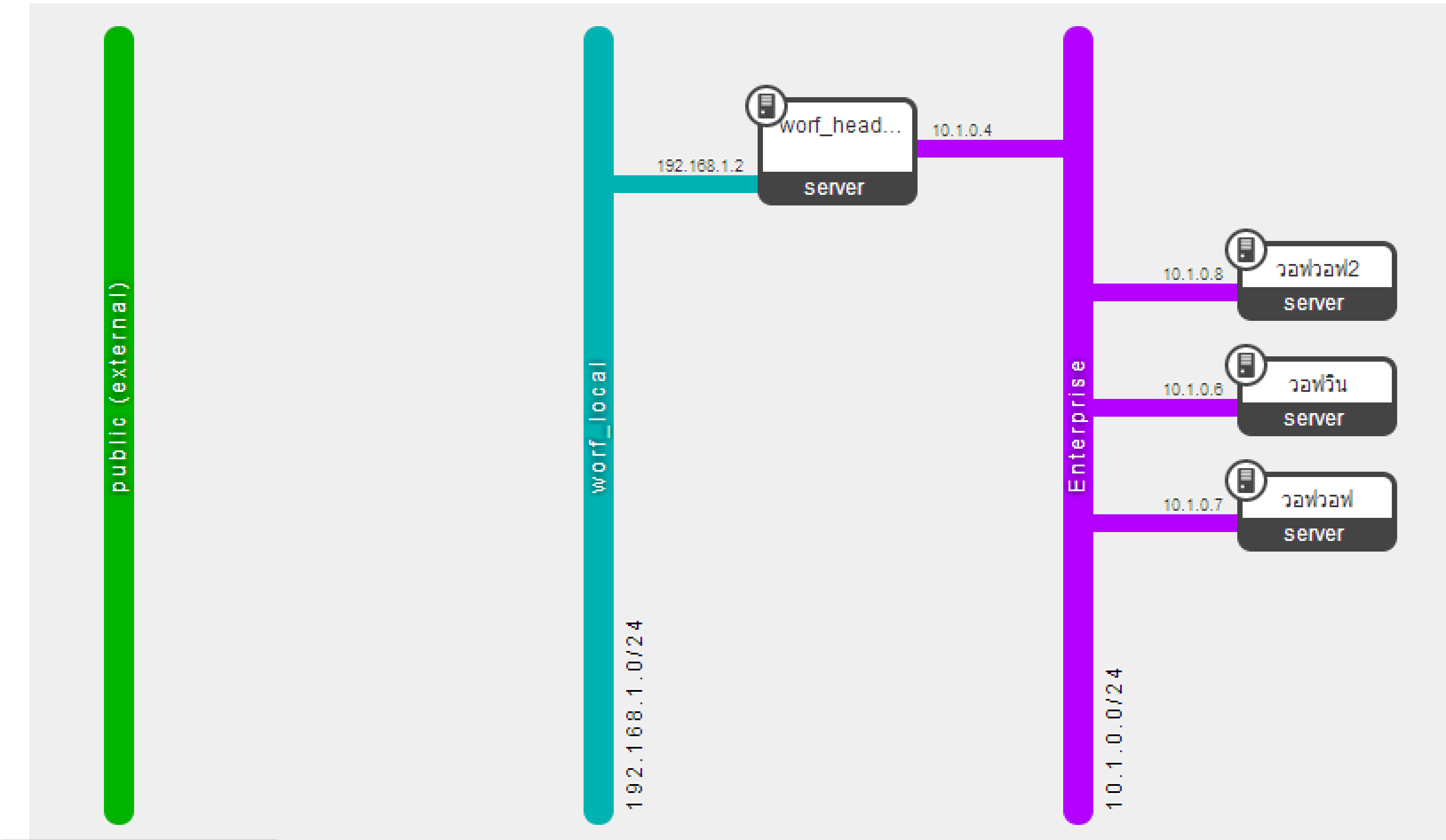

Launch Instance
Details
Access & Security
Networking
Volume Options
Post-Creation
Selected Networks
nic:1
Enterprise
nic:2
worf_local
Available networks
Choose network from Available networks to Selected Networks by push button or drag and drop, you may change nic order by drag and drop as well.
Cancel
Launch

Instances

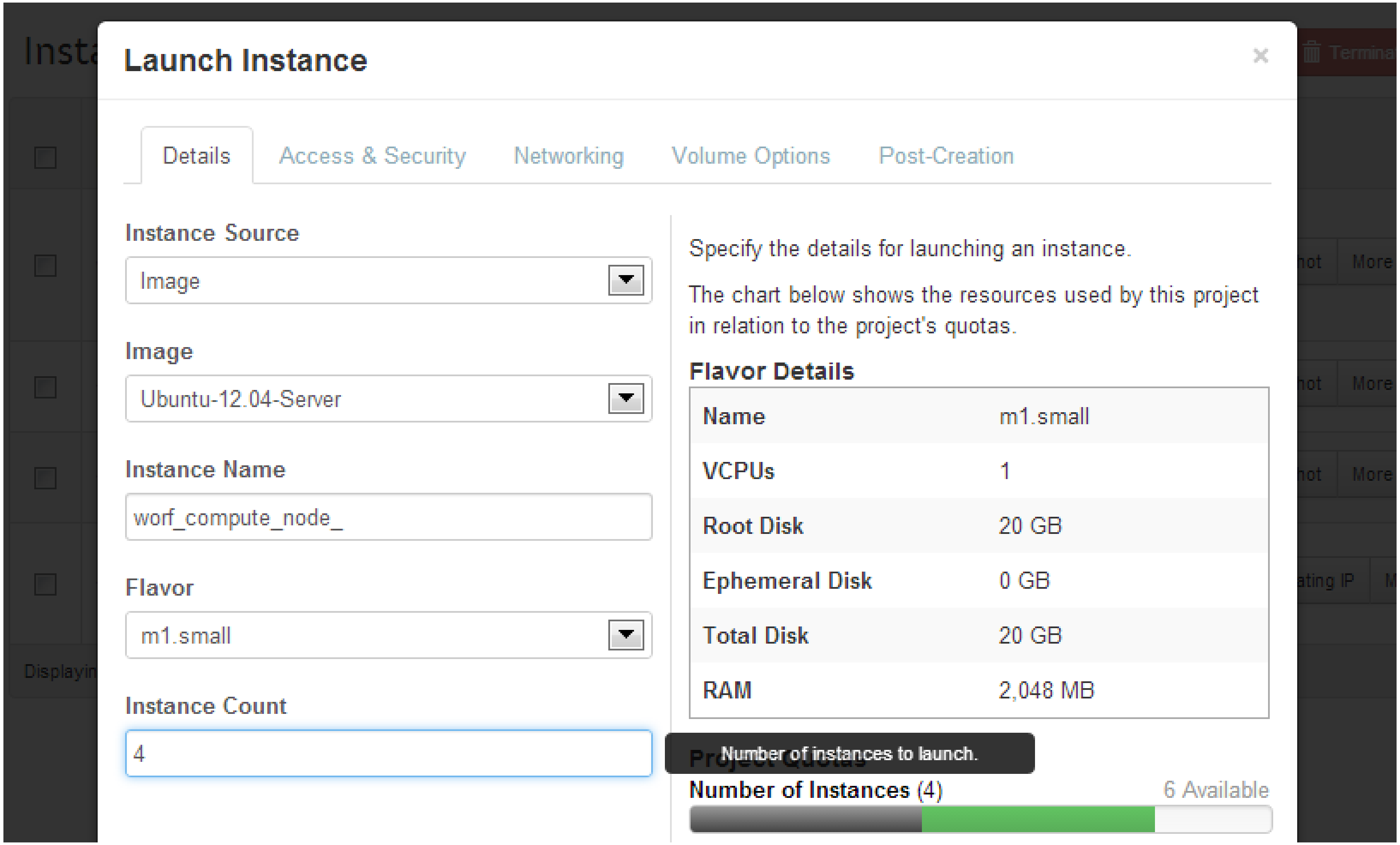
+ Launch Instance | Terminate Instances

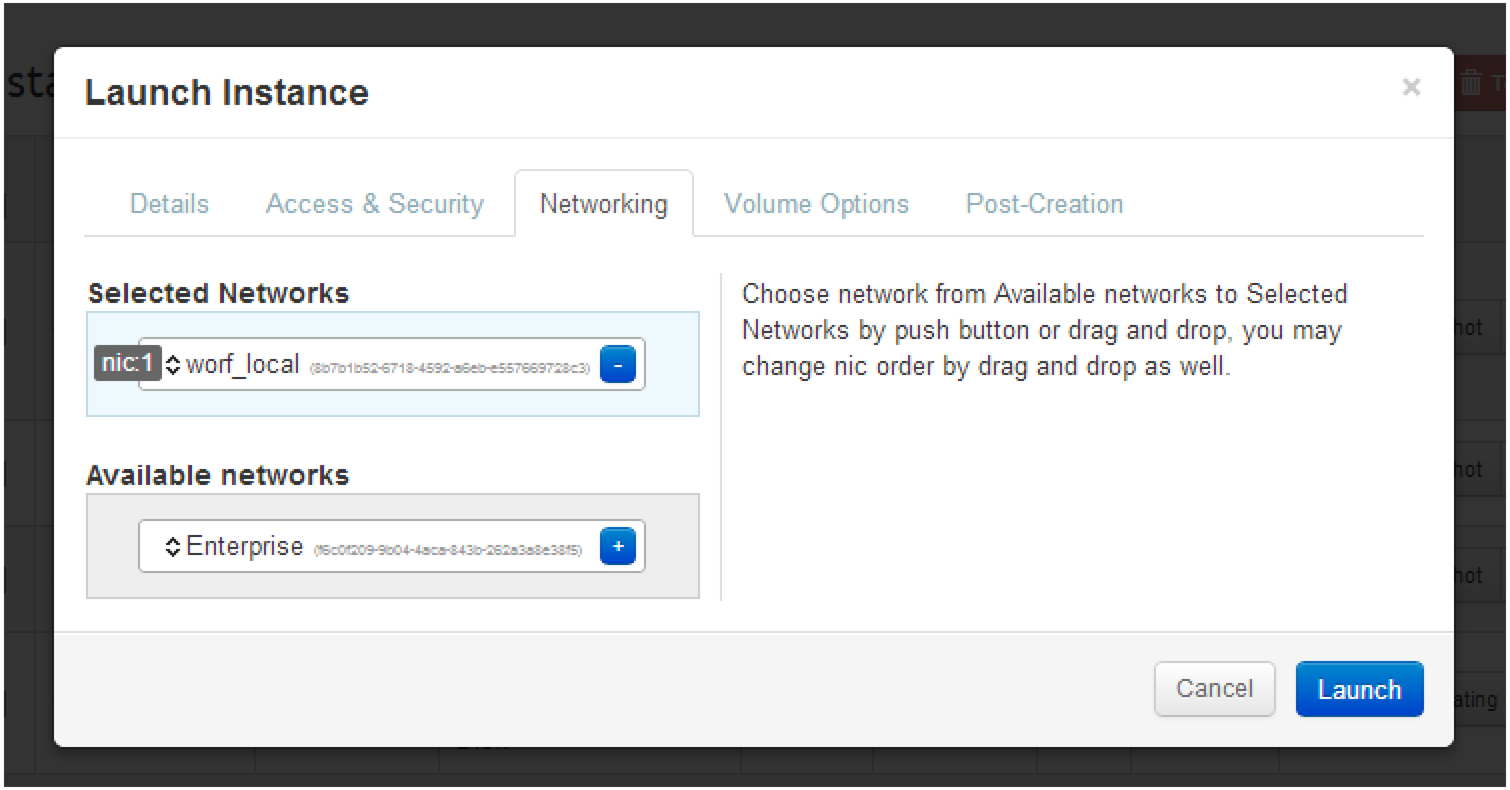
	Instance Name	IP Address	Size	Keypair	Status	Task	Power State	Actions
☐	worf_head_node	**worf_local** 192.168.1.2 **Enterprise** 10.1.0.4	medium | 4GB RAM | 2 VCPU | 40GB Disk	-	Active	None	Running	Create Snapshot More
☐	วอฟวอฟ2	10.1.0.8 10.100.20.19	medium | 4GB RAM | 2 VCPU | 40GB Disk	-	Active	None	Running	Create Snapshot More
☐	วอฟวิน	10.1.0.6 10.100.20.21	medium | 4GB RAM | 2 VCPU | 40GB Disk	-	Active	None	Running	Create Snapshot More
☐	วอฟวอฟ	10.1.0.7 10.100.20.20	m1.medium | 4GB RAM | 2 VCPU | 40GB Disk	-	Suspended	None	Shutdown	Associate Floating IP More

Displaying 4 items

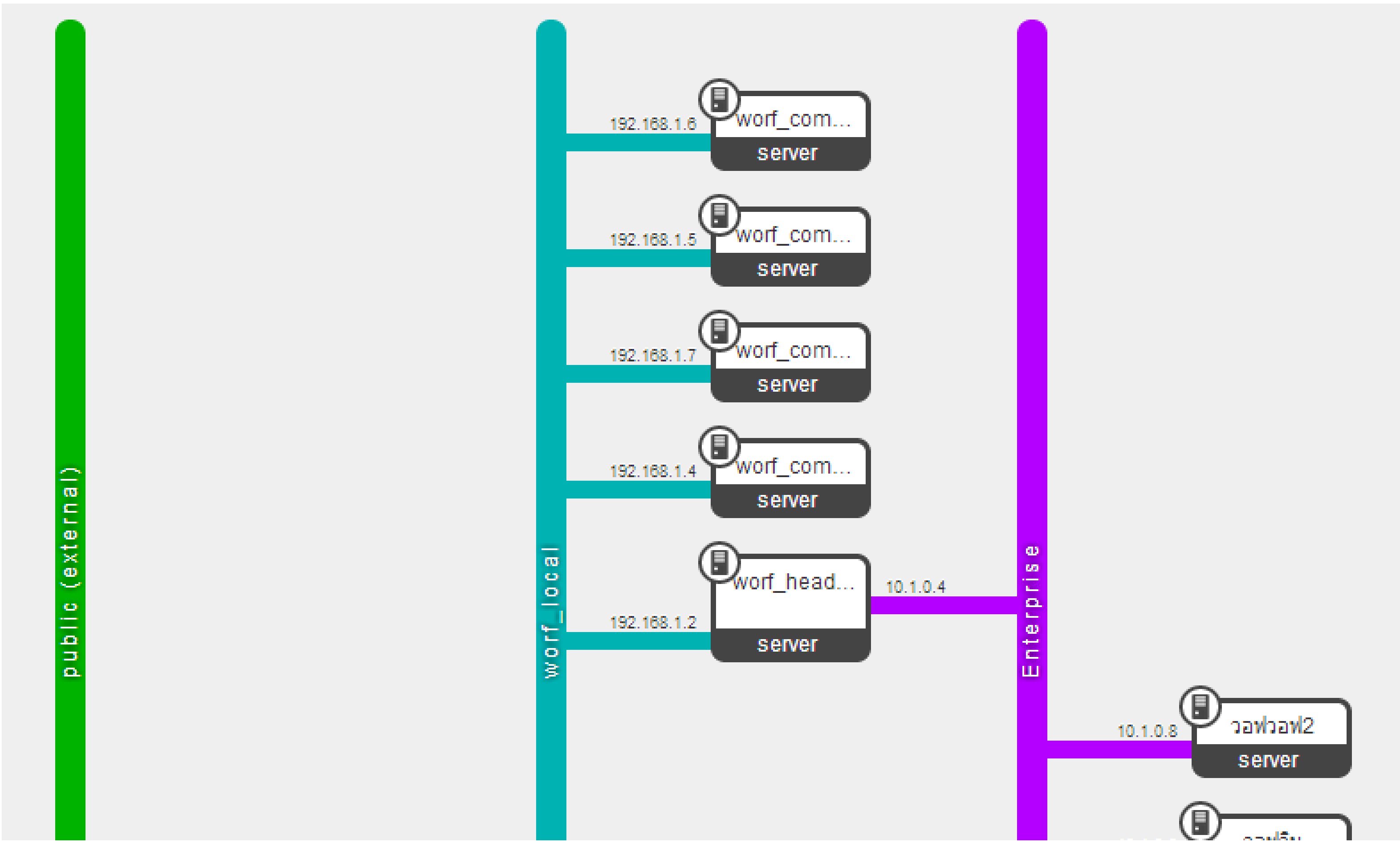

public (external)
worf_local
192.168.1.0/24
192.168.1.2
worf_head...
server
10.1.0.4
Enterprise
10.1.0.0/24
10.1.0.8
วอฟวอฟ2
server
10.1.0.6
วอฟวิน
server
10.1.0.7
วอฟวอฟ
server

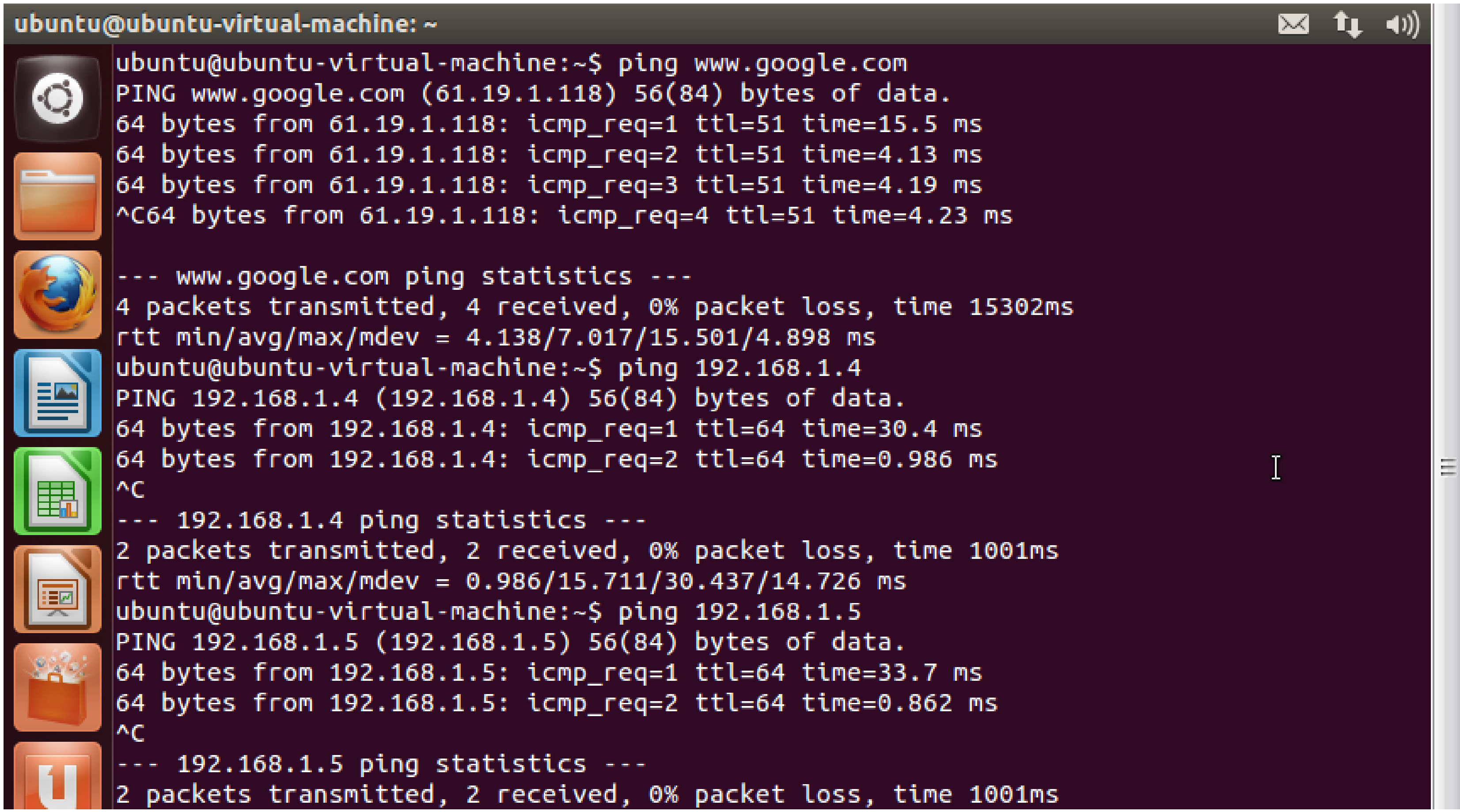
Launch compute nodes




Launch Instance
Details
Access & Security
Networking
Volume Options
Post-Creation
Selected Networks
nic:1
worf_local
Available networks
Enterprise
Choose network from Available networks to Selected Networks by push button or drag and drop, you may change nic order by drag and drop as well.
Cancel
Launch

	Instance Name	IP Address	Size	Keypair	Status	Task	Power State	Actions
	worf_compute_node_-49e4dc1d-8939-4dc2-9492-fe98abd2ac3d	192.168.1.6	m1.small | 2GB RAM | 1 VCPU | 20GB Disk	-	Active	None	Running	Create Snapshot More
	worf_compute_node_-e1a1023c-d1de-4c1f-91fb-5f0605047598	192.168.1.5	m1.small | 2GB RAM | 1 VCPU | 20GB Disk	-	Active	None	Running	Create Snapshot More
	worf_compute_node_-673ce66f-09a0-4eb1-aa6a-2be9756d52f0	192.168.1.7	m1.small | 2GB RAM | 1 VCPU | 20GB Disk	-	Active	None	Running	Create Snapshot More
	worf_compute_node_-ab4cb991-aeee-4b69-b0fd-9b82c903f592	192.168.1.4	m1.small | 2GB RAM | 1 VCPU | 20GB Disk	-	Active	None	Running	Create Snapshot More
	worf_head_node	**worf_local** 192.168.1.2 **Enterprise** 10.1.0.4	medium | 4GB RAM | 2 VCPU | 40GB Disk	-	Active	None	Running	Create Snapshot More


public (external)
worf_local
Enterprise
192.168.1.6
worf_com...
server
192.168.1.5
worf_com...
server
192.168.1.7
worf_com...
server
192.168.1.4
worf_com...
server
192.168.1.2
worf_head...
server
10.1.0.4
10.1.0.8
วอฟวอฟ2
server


ubuntu@ubuntu-virtual-machine: ~
ubuntu@ubuntu-virtual-machine:~$ ping www.google.com
PING www.google.com (61.19.1.118) 56(84) bytes of data.
64 bytes from 61.19.1.118: icmp_req=1 ttl=51 time=15.5 ms
64 bytes from 61.19.1.118: icmp_req=2 ttl=51 time=4.13 ms
64 bytes from 61.19.1.118: icmp_req=3 ttl=51 time=4.19 ms
^C64 bytes from 61.19.1.118: icmp_req=4 ttl=51 time=4.23 ms
--- www.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 15302ms
rtt min/avg/max/mdev = 4.138/7.017/15.501/4.898 ms
ubuntu@ubuntu-virtual-machine:~$ ping 192.168.1.4
PING 192.168.1.4 (192.168.1.4) 56(84) bytes of data.
64 bytes from 192.168.1.4: icmp_req=1 ttl=64 time=30.4 ms
64 bytes from 192.168.1.4: icmp_req=2 ttl=64 time=0.986 ms
^C
--- 192.168.1.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.986/15.711/30.437/14.726 ms
ubuntu@ubuntu-virtual-machine:~$ ping 192.168.1.5
PING 192.168.1.5 (192.168.1.5) 56(84) bytes of data.
64 bytes from 192.168.1.5: icmp_req=1 ttl=64 time=33.7 ms
64 bytes from 192.168.1.5: icmp_req=2 ttl=64 time=0.862 ms
^C
--- 192.168.1.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms