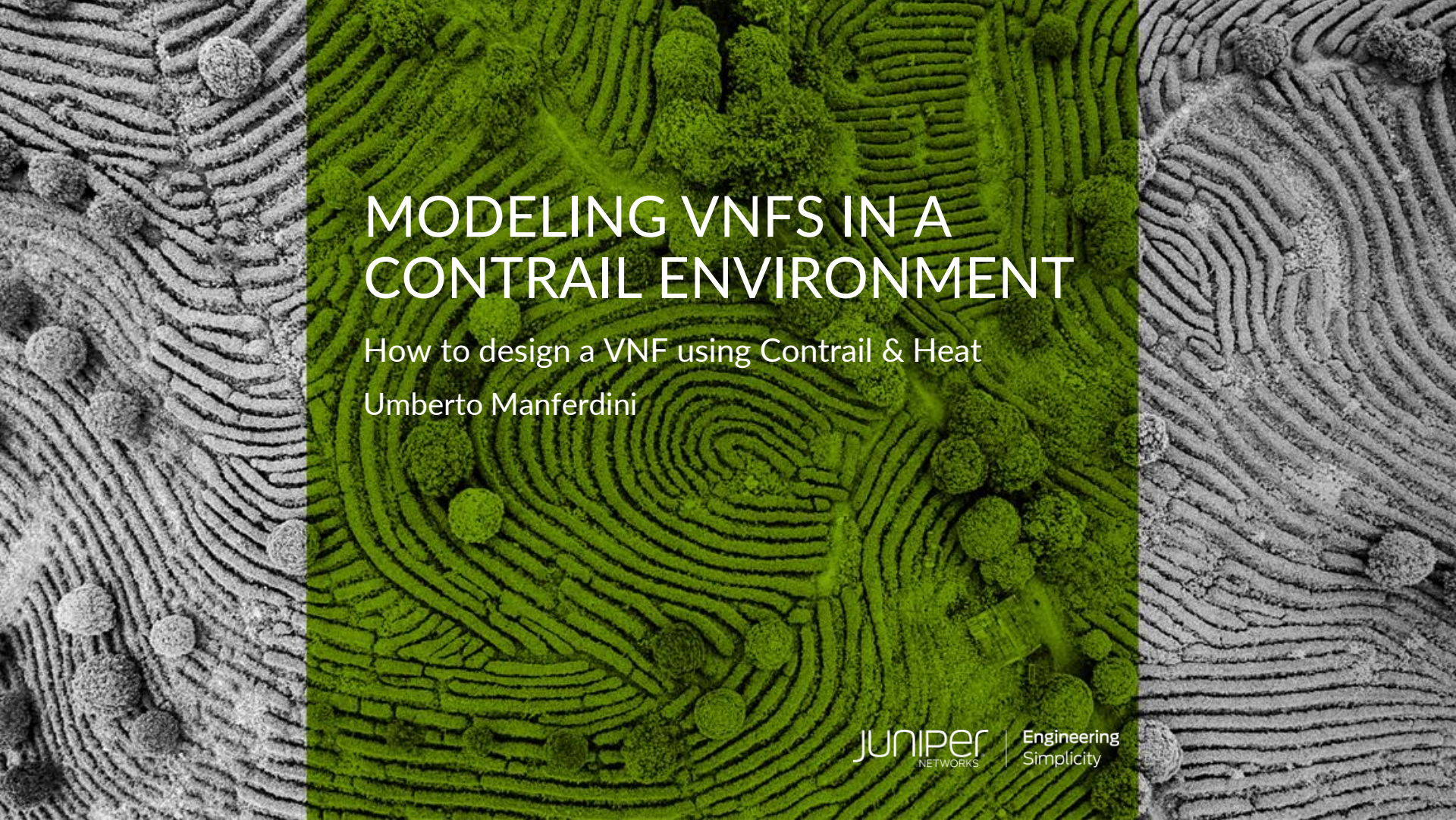

MODELING VNFS IN A CONTRAIL ENVIRONMENT
How to design a VNF using Contrail & Heat
Umberto Manferdini
JUNIPER NETWORKS
Engineering Simplicity

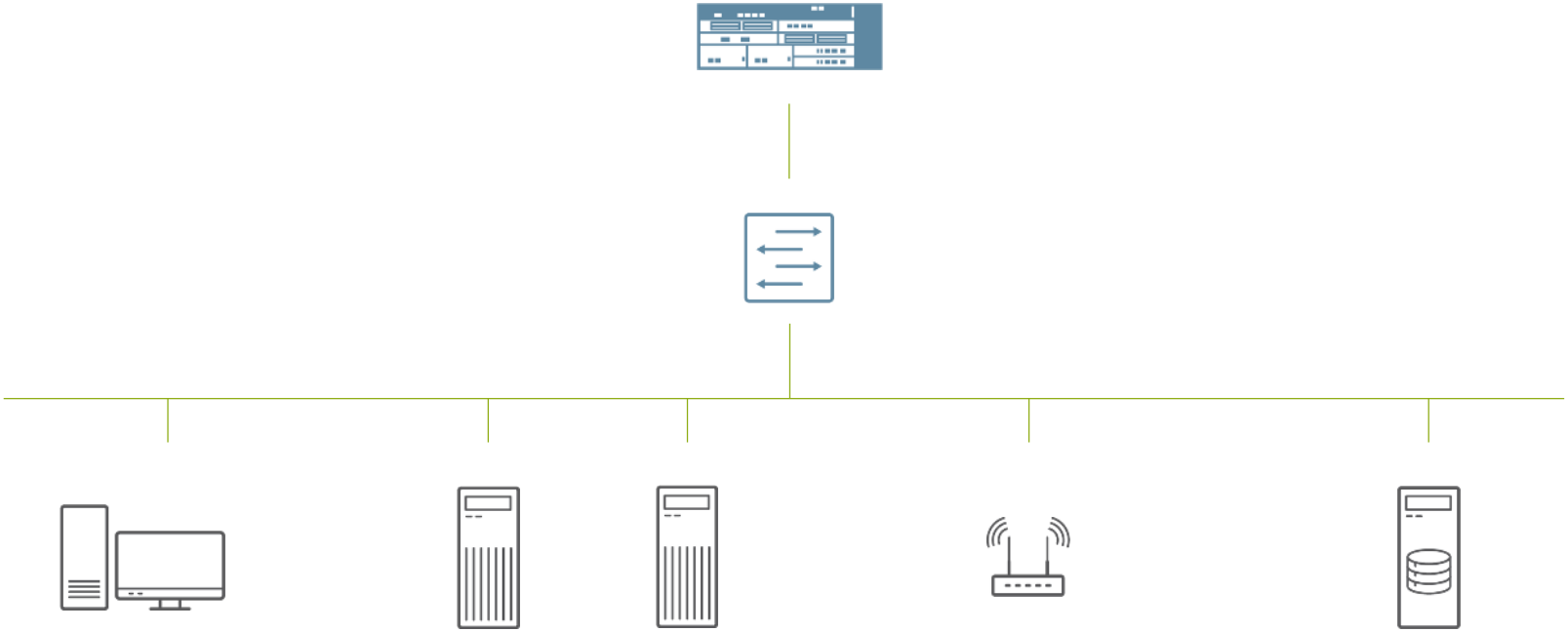
AGENDA

- Where are we now?
- What's Heat?
- Contrail and Heat: modeling VNFs
- From vanilla Neutron to Juniper Contrail
- Service Chaining

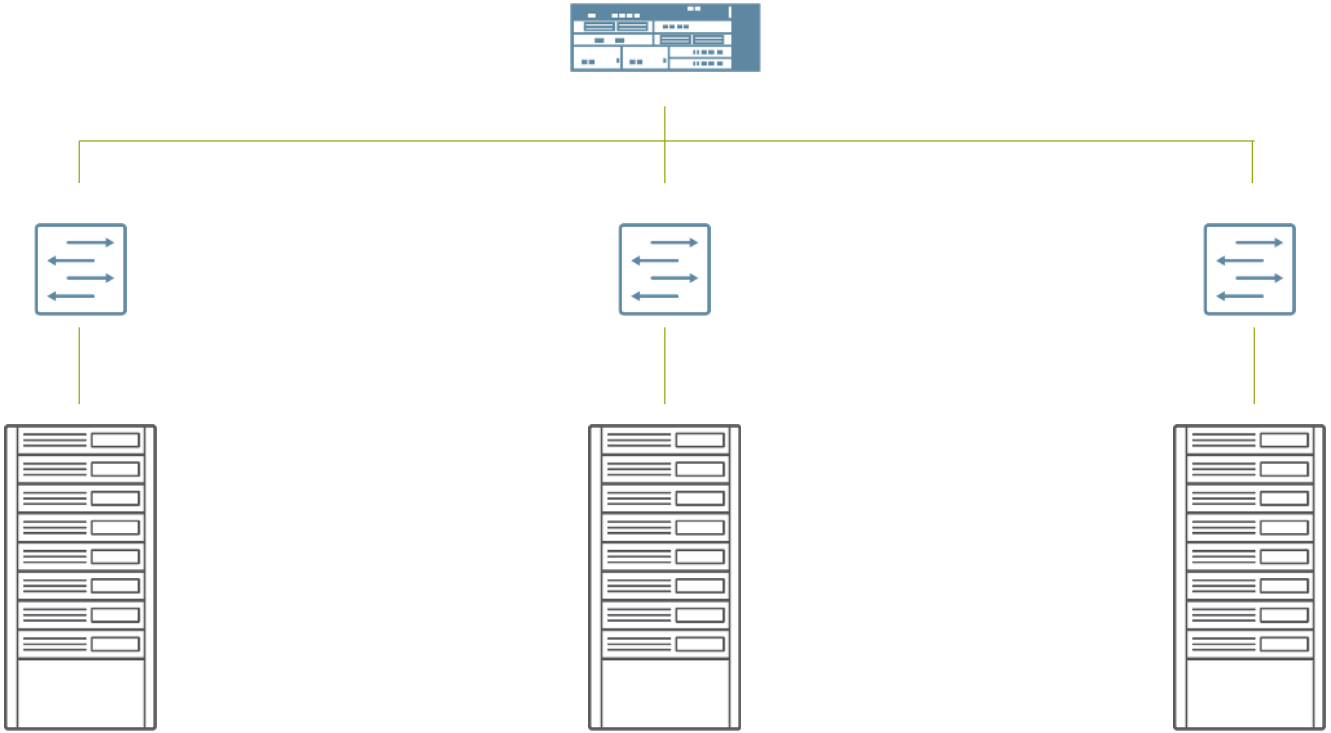
JUNIPER CONTRAIL

- Juniper SDN Controller
- Manages virtual networking within the DC
- Based on well known protocols and solutions like BGP and VPNs

ORIGINALLY IT WAS A SWITCHED NETWORK

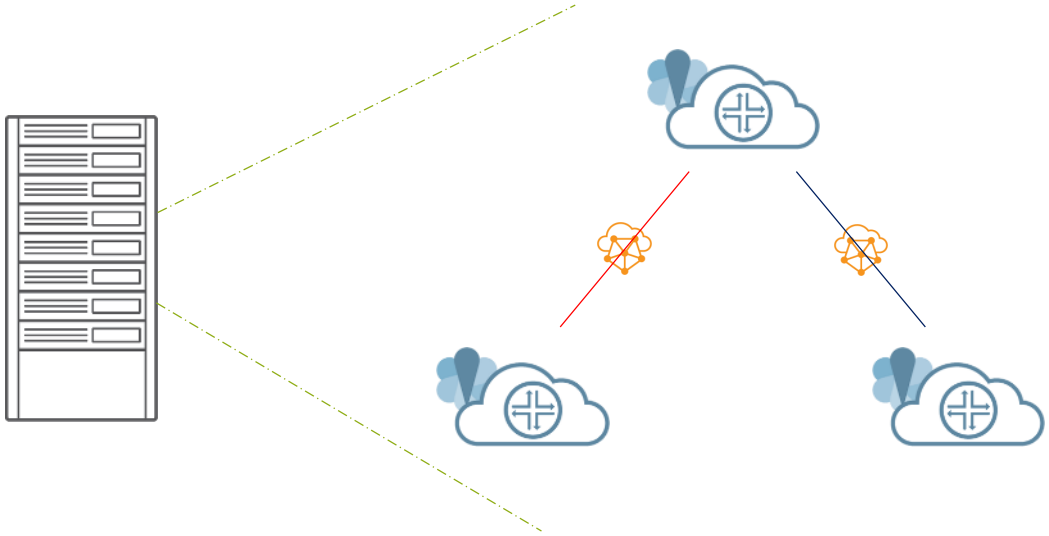


THEN DATA CENTERS CAME



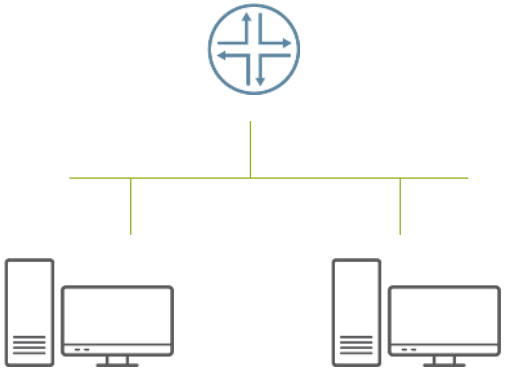
FINALLY VIRTUALIZATION APPEARED

- A network within a network
- A network inside a server
- Virtual Machines are the new appliances
- Virtual Networks are the new switched networks

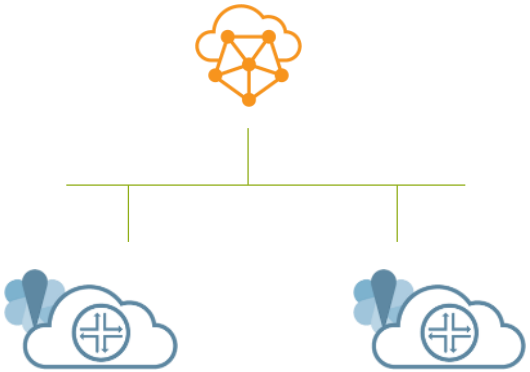


SAME THING…DIFFERENT OBJECTS

physical switch/router



virtual router

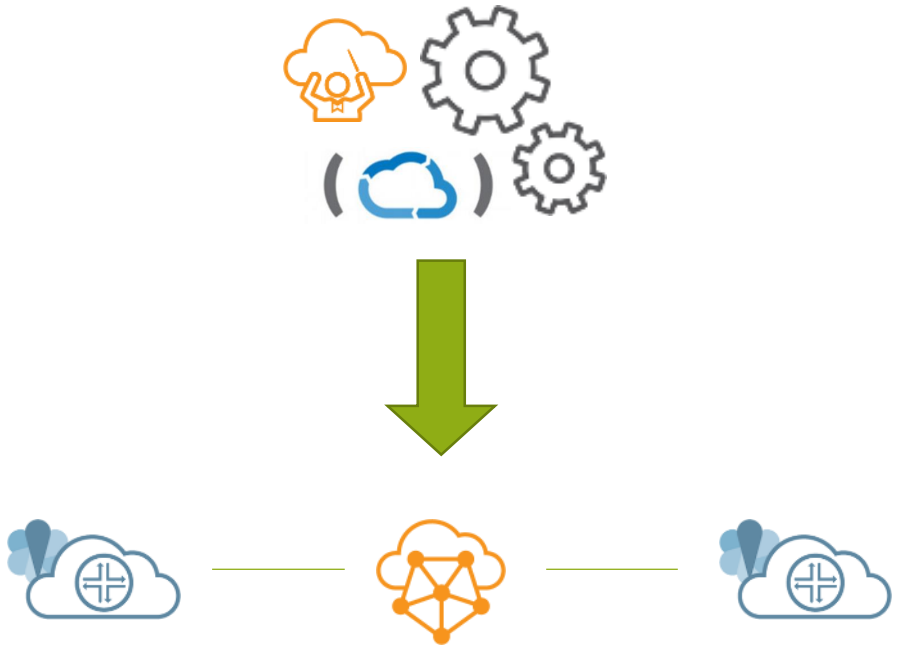


CALL FOR NEW MODELS

- Yesterday, we used cables to connect physical devices
- Now, we have to connect virtual machines
- We need new models to describe these new scenarios

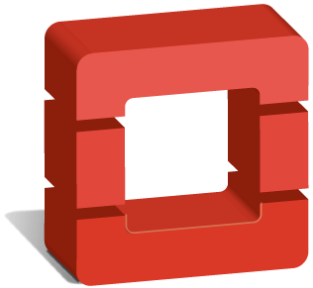
VIRTUALIZATION BRINGS AUTOMATION

- Virtualization is software
- Software is more inclined to be automated
- Virtualization opens to network automation



AUTOMATION MEANS HEAT

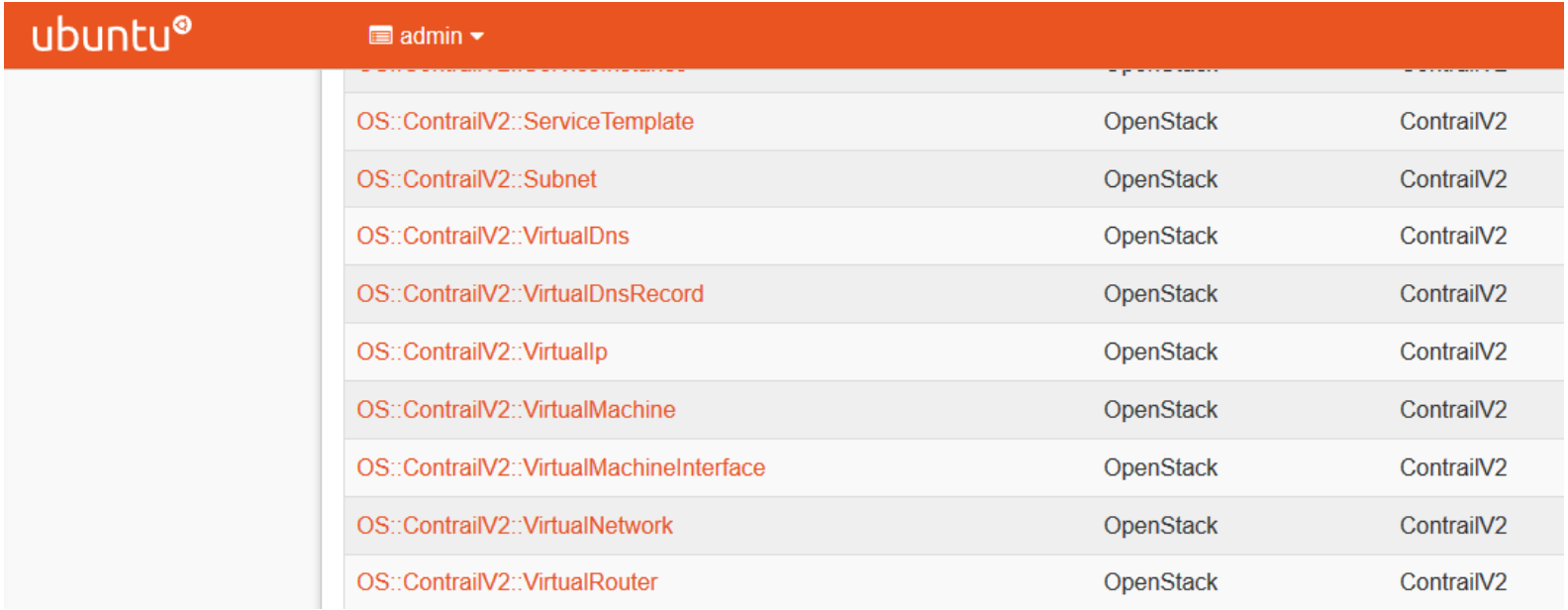
- Openstack deploys and manage the DC
- Openstack has tool for automation called Heat
- Contrail fully integrates with Heat
- Contrail provides its own Heat objects



AND HEAT MEANS YAML

- Heat is a built-in Openstack module
- Heat uses YAML language
- Easy to read and write

CONTRAIL HAS ITS OWN OBJECTS



AUTOMATION THROUGH STACKS

- Heat creates stacks
- Stacks are a list resources
- Resources are virtual network, virtual machines, etc…

WHAT DO I HAVE TO DO?

New Virtual Machine on a Virtual Network

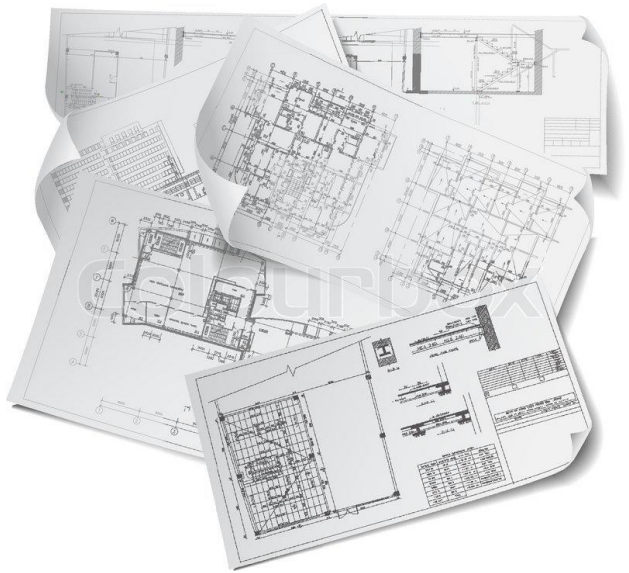
New Virtual Network

Manage VLANS in a “v” environment

Extend Virtual Network outside the DC

FROM WORDS TO HEAT

- Need to take an abstract input as request
- For example “configure a virtual network”
- And translate it into a Heat template

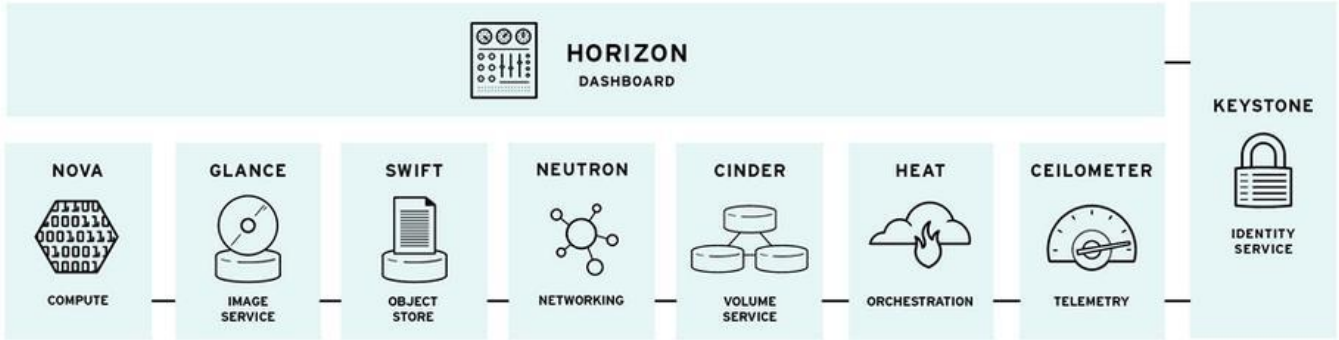


WHAT'S A HEAT TEMPLATE?

- Structured YAML files
- Template file, where we define resources
- Environment file, where we assign values to parameters

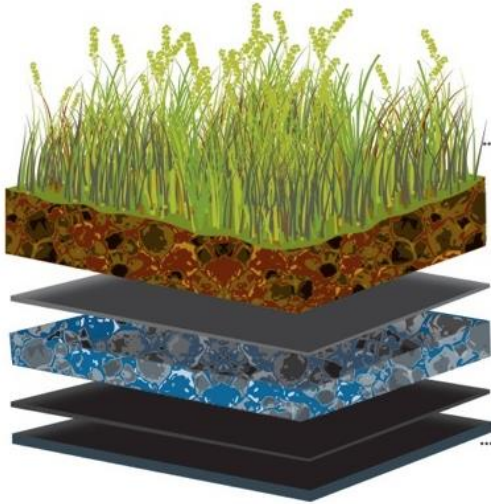
OBJECTS FOR EVERYONE

- Openstack offers a wide range of Heat object
- Nova objects
- Glance objects
- Contrail objects
- …



INSIDE A HEAT TEMPLATE

- Heat templates made of multiple sections
- Two main sections
- Parameters section
- Resource section



DECLARING PARAMETERS

```
heat_template_version: 2015-04-30

parameters:
  myname :
     type: string
     default: 'nameX'
  myimage :
     type: string
     default: 'ubuntu'
  myflavor :
     type: string
     default: 'smallvm'
```

DECLARING RESOURCES

```
resources:

test-net:
    type: OS::Neutron::Net
    properties:
        name: 'heat-net'

test-subnet:
  type: OS::Neutron::Subnet
  properties:
    name: 'heat-subnet'
    network_id: { get_resource: test-net }
    cidr: '192.168.1.0/16'

server1:
  type: OS::Nova::Server
  properties:
    name: { get_param: myname }
    image: { get_param: myimage }
    flavor: { get_param: myflavor }
    networks:
      - network: { get_resource: test-net }
```

resource type

hardocded value

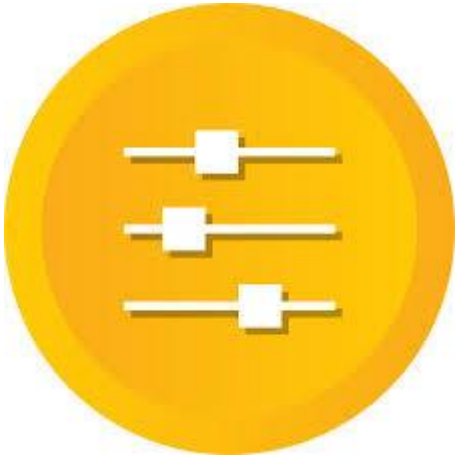
get_resource

get_param

ENVIRONMENT FILE

- Parameters defined in the template file
- We need to give them a value
- Environment file is for this!

```
parameters:
  myname : Ubuntu-1
  myimg : ubu14-iso
  myflavor : linux-perf
```



WHO'S FIRST?

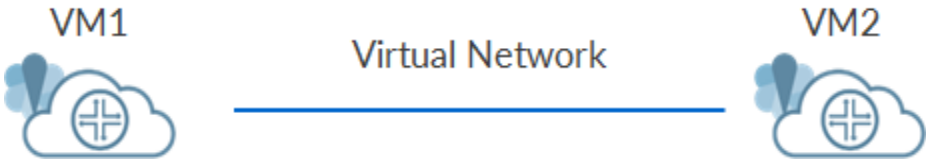
- Heat is smart
- But not too smart
- Some order is required
- Use "depends_on" to manage it


```
server1:
  type: OS::Nova::Server
  depends_on [test-subnet, test-net]
  properties:
    name: { get_param: myname }
    image: { get_param: myimage }
    flavor: { get_param: myflavor }
    networks:
      - network: { get_resource: test-net }
```

- Virtual network
- Virtual machine interface
- Instance IP
- BGPaaS
- Sub-interfaces
- Allowed address pair

CONTRAIL VIRTUAL NETWORKS

- Key element in Contrail networking
- Used to connect Virtual Machines



VIRTUAL NETWORK PROPERTIES

- Virtual Networks has a name
- Reference an IPAM (Contrail wrapper for virtual networks)


```
resources:
  vip_vn:
    type: OS::ContrailV2::VirtualNetwork
    properties:
      name: ‘myvn’
      network_ipam_refs: ['default-domain:default-project:default-network-ipam']
```

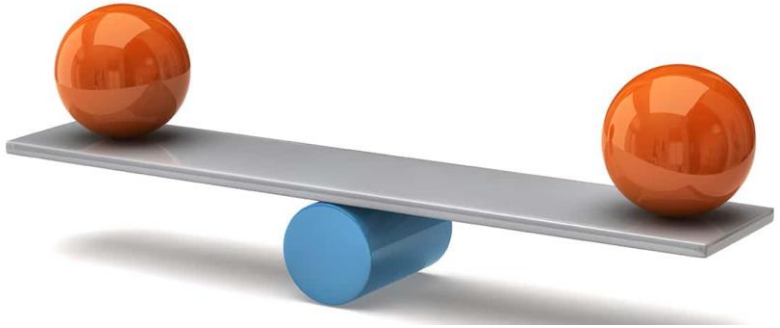
CUSTOMIZING THE VIRTUAL NETWORK

- Forwarding mode
- RPF check
- Flooding unknown unicast
- VXLAN identifier

```
resources:
  vip_vn:
    type: OS::ContrailV2::VirtualNetwork
    properties:
      virtual_network_properties:
        {
          virtual_network_properties_forwarding_mode: ‘l3’,
          virtual_network_properties_rpf: ‘enable’,
          virtual_network_properties_vxlan_network_identifier: ‘1234’,
        }
      flood_unknown_unicast: ‘true’
```

HOW DO I ECMP?

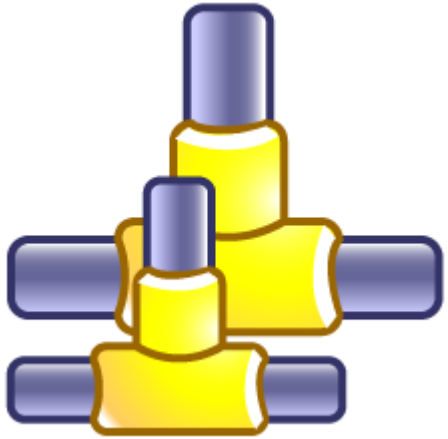
- Contrail supports ECMP
- Up to 5 configurable fields



```
resources:
  vip_vn:
    type: OS::ContrailV2::VirtualNetwork
    properties:
      ecmp_hashing_include_fields:
        {
          ecmp_hashing_include_fields_destination_ip: ‘true’,
          ecmp_hashing_include_fields_destination_port: ‘false’,
          ecmp_hashing_include_fields_hashing_configured: ‘true’,
          ecmp_hashing_include_fields_ip_protocol: ‘true’,
          ecmp_hashing_include_fields_source_ip: ‘false’,
          ecmp_hashing_include_fields_source_port: ‘false’,
        }
      port_security_enabled: ‘true’
```

CREATING SUBNETS

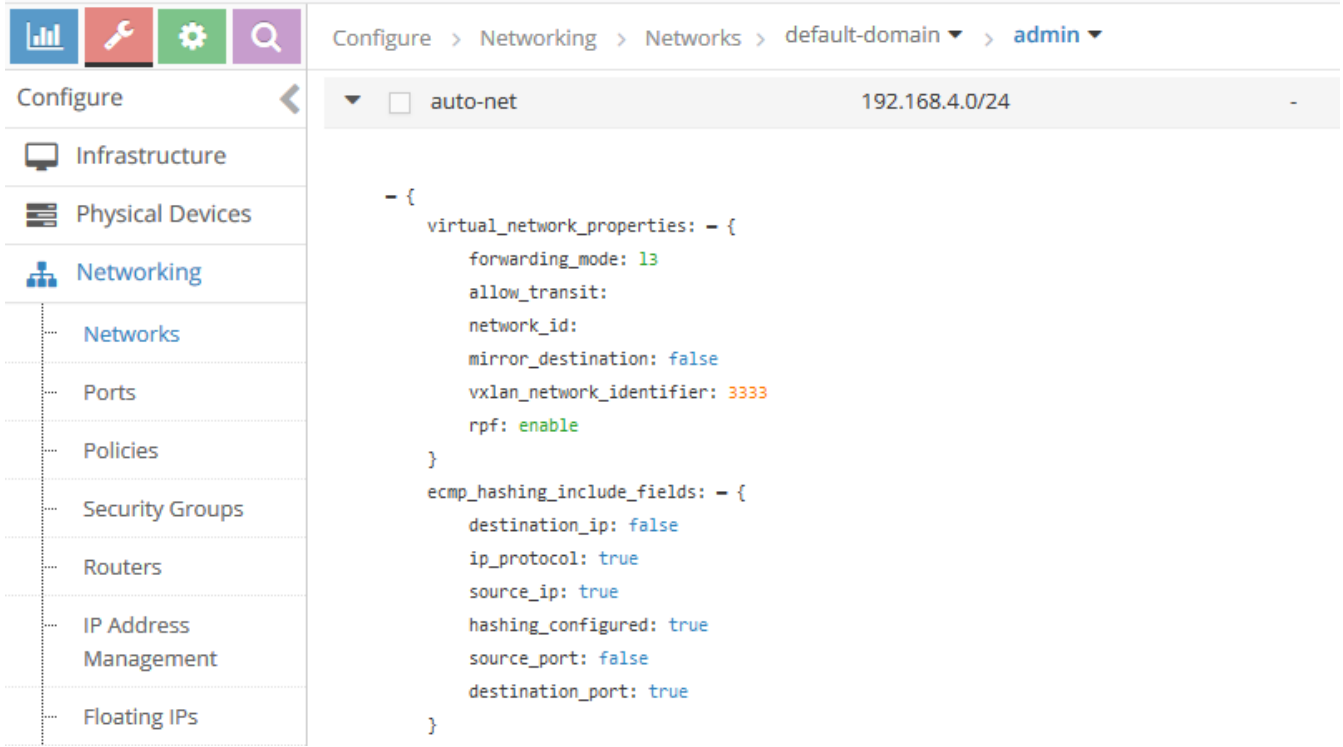
- Not a dedicated object
- Network address and mask
- Allocation pools
- DHCP
- Gateway
- Allocation order



SUBNETS IN HEAT

```
resources:
  vip_vn:
    type: OS::ContrailV2::VirtualNetwork
    properties:
      network_ipam_refs_data:
        [{
          network_ipam_refs_data_ipam_subnets:
            [{
              network_ipam_refs_data_ipam_subnets_subnet:
                {
                  network_ipam_refs_data_ipam_subnets_subnet_ip_prefix: '192.168.33.0',
              network_ipam_refs_data_ipam_subnets_subnet_ip_prefix_len: '24',
                },
                network_ipam_refs_data_ipam_subnets_addr_from_start: 'true',
                network_ipam_refs_data_ipam_subnets_default_gateway: '192.168.33.3',
                network_ipam_refs_data_ipam_subnets_enable_dhcp: 'false',
                network_ipam_refs_data_ipam_subnets_allocation_pools:
                  [{
                    network_ipam_refs_data_ipam_subnets_allocation_pools_start: '192.168.33.5',
                    network_ipam_refs_data_ipam_subnets_allocation_pools_end: '192.168.33.10',
                  }]
         }]
        }]
```

CHECKING EVERYTHING IS ALRIGHT

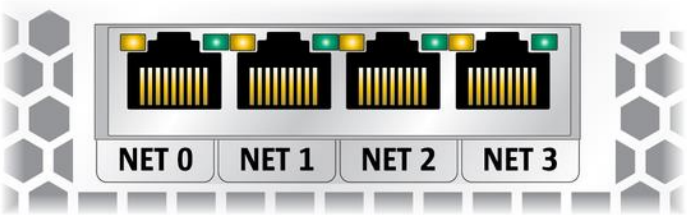


VIRTUAL NIC

- Virtual machine port
- Connects VM to the Virtual Network
- It is actually 2 objects
- Virtual Machine Interface
- Instance IP

VIRTUAL MACHINE INTERFACE

- Minimal configuration
- Name
- Virtual Network Reference




```
resources:
  vmi_newnet:
    type: OS::ContrailV2::VirtualMachineInterface
    properties:
      name: 'my_vmi'
      virtual_network_refs: [{ list_join: [':', { get_attr: [ newnet_vn, fq_name ] } ] }]
```

Internal function
Fully Qualified Name
prepends “domain:project”

LEVERAGING EXISTING NETWORKS

- Existing network
- Simply pass the FQDN network name as a parameter
- FQDN name means “domain:project:network”

```
vmi_prex:
   type: OS::ContrailV2::VirtualMachineInterface
   properties:
     virtual_network_refs: [ {get_param: mgmt-net} ]
```

```
parameters:
  mgmt-net: 'default-domain:admin:mgmt'
```

env file

I WANT MY OWN MAC

- By default, Contrail automatically assigns a MAC to a port
- Optionally, we can assign a custom MAC


```
resources:
  vmi_newnet:
    type: OS::Contrailv2::VirtualMachineInterface
    properties:
      virtual_machine_interface_mac_addresses:
        {
            virtual_machine_interface_mac_addresses_mac_address: [ ‘aa:bb:cc:dd:ee:ff’],
        }
```

A PORT NEEDS AN IP

- Second element is the Instance IP
- References a virtual networks
- References a VMI


```
instanceip_1:
  type: OS::ContrailV2::InstanceIp
  depends_on: [ vmi_newnet, newnet_vn ]
  properties:
   virtual_machine_interface_refs: [{ get_resource: vmi_newnet }]
   virtual_network_refs: [{ list_join: [':', { get_attr: [ newnet_vn, fq_name ] } ] }]
```

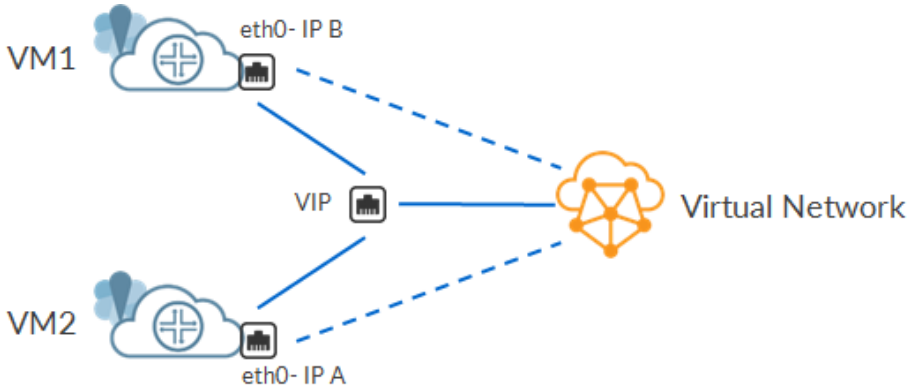
MY IP OR YOUR IP?

- We can reference an existing virtual network
- Or specify a fixed IP address

```
iip_fixed:
   type: OS::ContrailV2::InstanceIp
   depends_on: [ vmi_obj ]
   properties:
     virtual_machine_interface_refs: [{ get_resource: vmi_obj }]
     virtual_network_refs: [{ get_param: mgmt-net }]
     instance_ip_address: ‘192.168.1.4’
```

VIP ADDRESS

- Many VMs sharing an IP
- To spread the load on multiple devices
- Leveraging Contrail ECMP capabilities



ALLOWING MULTIPLE ADDRESSES ON A PORT

192.168.1.13

NO

192.168.1.14

192.168.1.14

YES

192.168.1.14

VIP MEANS ECMP

- VIP configured on the VMI
- Need to allow an Allowed Address Pair
- Routes automatically added in Contrail
- Multiple routes towards the same VIP are seen as ECMP paths

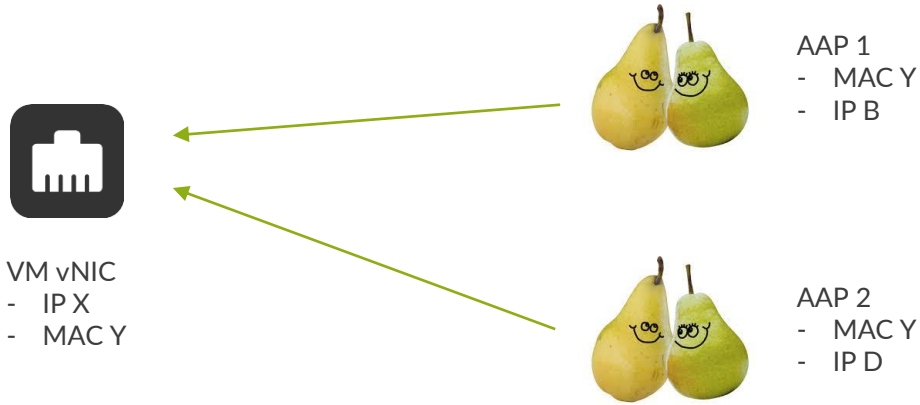
Prefix: 192.168.5.100 / 32 (4 Routes)	
ECMP Composite sub nh count: 2	Source IP: Destination IP: vrf: Ref count: Policy: enabled Peer: 192.168.100.2 Valid: true Label: -1
ECMP Composite sub nh count: 2	Source IP: Destination IP: vrf: Ref count: Policy: enabled Peer: Ecmp Valid: true Label: 20
interface	Interface: tap24c1f7ca-fe Destination VN: default-domain:admin:vmi-net Policy: enabled Peer: LocalVmPort Valid: true
interface	Interface: tapcfed0090-ec Destination VN: default-domain:admin:vmi-net Policy: enabled Peer: LocalVmPort Valid: true

ALLOWED ADDRESS PAIR

```
resources:
  vmi_1:
    type: OS::ContrailV2::VirtualMachineInterface
    properties:
      name: 'my_vmi'
      virtual_network_refs: [{ list_join: [':', { get_attr: [ new_vn, fq_name ] } ] }]
      virtual_machine_interface_mac_addresses:
        {
            virtual_machine_interface_mac_addresses_mac_address: ‘aa:bb:cc:dd:ee:ff’,
        }
      virtual_machine_interface_allowed_address_pairs:
        {
         virtual_machine_interface_allowed_address_pairs_allowed_address_pair: [{
          virtual_machine_interface_allowed_address_pairs_allowed_address_pair_ip:
                {
           virtual_machine_interface_allowed_address_pairs_allowed_address_pair_ip_ip_prefix: '192.168.1.100’,
           virtual_machine_interface_allowed_address_pairs_allowed_address_pair_ip_ip_prefix_len: '32',
                },
              virtual_machine_interface_allowed_address_pairs_allowed_address_pair_mac: ‘aa:bb:cc:dd:ee:ff’,
              virtual_machine_interface_allowed_address_pairs_allowed_address_pair_address_mode: 'active-active',
          }]
        }
```

MULTIPLE IPS ON THE SAME MAC

- Configuration within the VMI object
- Add allowed pairs
  - MAC address
  - IP address

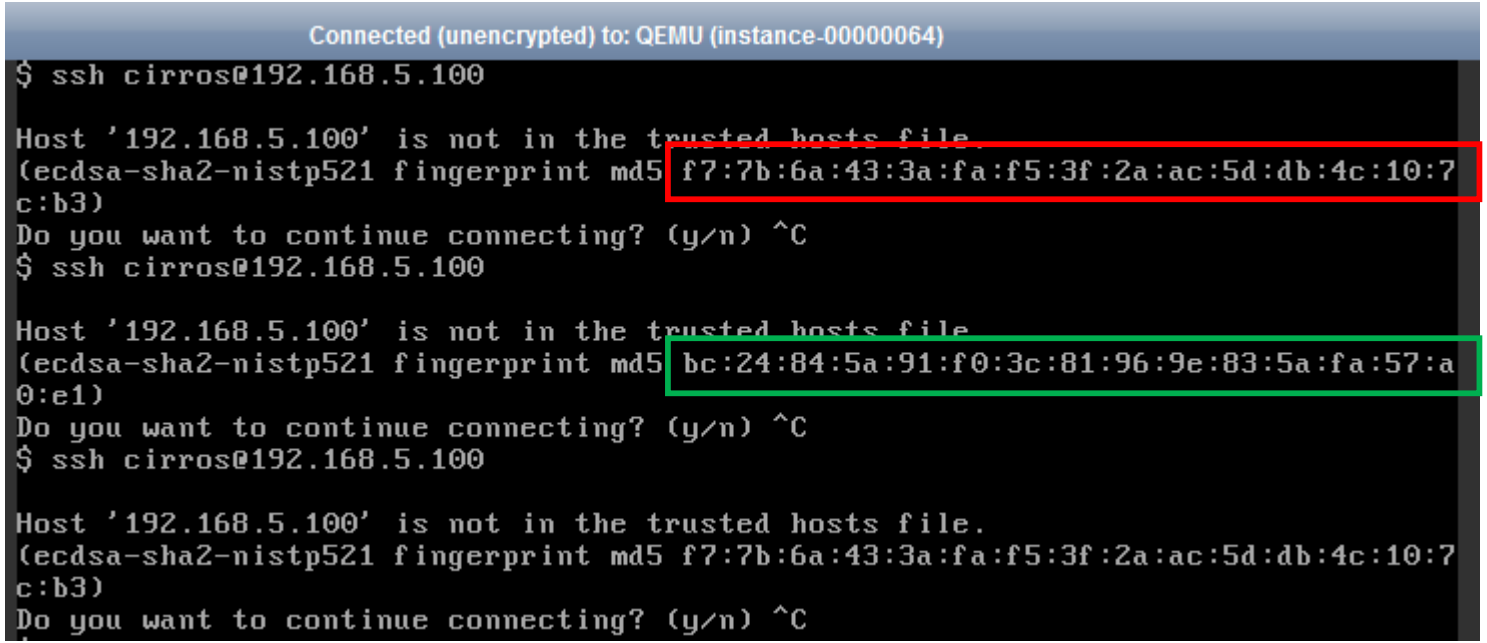


CONTRAIL PICKS THE MAC

- We can avoid to define a custom MAC
- Just remove MAC definition and AAP MAC reference

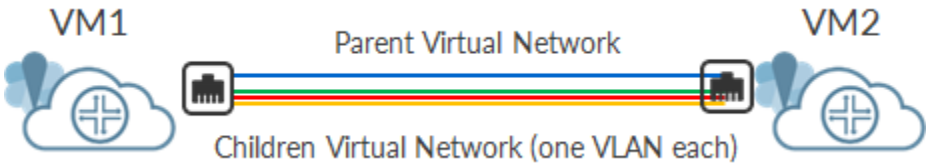

```
resources:
  vmi_1:
    type: OS::ContrailV2::VirtualMachineInterface
    properties:
      name: 'my_vmi'
      virtual_network_refs: [{ list_join: [':', { get_attr: [ new_vn, fq_name ] } ] }]
      virtual_machine_interface_allowed_address_pairs:
        {
          virtual_machine_interface_allowed_address_pairs_allowed_address_pair: [{
              virtual_machine_interface_allowed_address_pairs_allowed_address_pair_ip:
                {
                  virtual_machine_interface_allowed_address_pairs_allowed_address_pair_ip_ip_prefix: { get_param: '192.168.1.100' },
                  virtual_machine_interface_allowed_address_pairs_allowed_address_pair_ip_ip_prefix_len: { get_param: '32' },
                },
          virtual_machine_interface_allowed_address_pairs_allowed_address_pair_address_mode: { get_param: 'active-active' },
          }]
        }
```

VIP IN ACTION



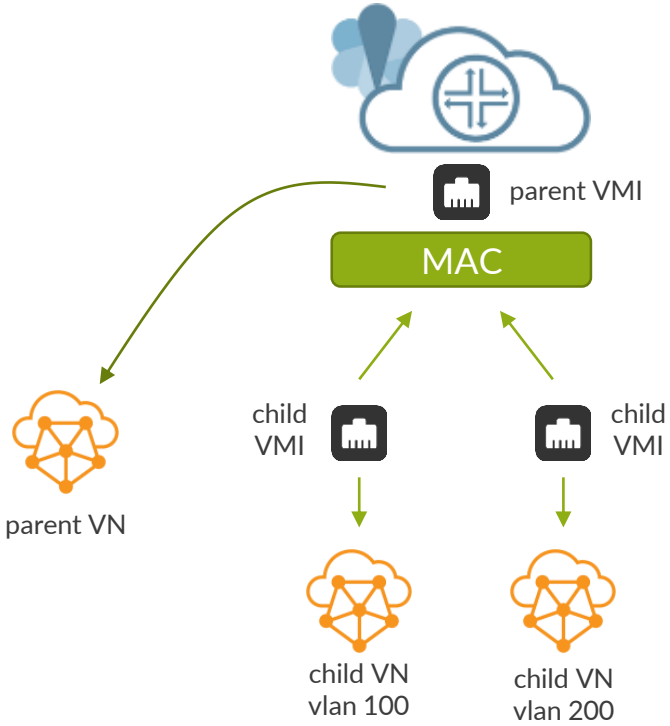
TRUNK PORTS

- Multiple vlans inside the VM
- Virtual NIC must become a Trunk
- Do not Use Openstack Trunk Object
- Use Contrail sub-interfaces (which came earlier!)



HOW DO TRUNKS WORK?

- Concept is easy
- Create a parent network
- Attach a VMI to it
- Create child virtual networks, one for each vlan
- Create child VMIs, one for each vlans
- Child VMIs must reference the parent VMI
- Child VMIs must use parent VMI MAC address



FIRST THERE IS A FATHER

```
resources:
  parent_vn:
    type: OS::ContrailV2::VirtualNetwork
    properties:
      name: { get_param: parent }
      network_ipam_refs: ['default-domain:my-project:default-ipam']
      network_ipam_refs_data:
        [{
          network_ipam_refs_data_ipam_subnets:
            [{
              network_ipam_refs_data_ipam_subnets_subnet:
                {
                  network_ipam_refs_data_ipam_subnets_subnet_ip_prefix: '172.30.1.0',
                  network_ipam_refs_data_ipam_subnets_subnet_ip_prefix_len: '24',
                },
          }]
     }]

vm_1:
    type: OS::Nova::Server
    properties:
      name: 'trunk_vm'
      image: 'vsrx'
      flavor: 'vsrx'
      networks:
        - network: { get_resource: parent_vn }
```

parent VMI

THEN THE CHILDREN

- Child VN is a standard VN
- No vlan defined here
- No link to parent

```
child_vn_1:
    type: OS::ContrailV2::VirtualNetwork
    properties:
      name: 'child1'
      network_ipam_refs: ['default-domain:my-project:default-ipam']
      network_ipam_refs_data:
        [{
          network_ipam_refs_data_ipam_subnets:
            [{
              network_ipam_refs_data_ipam_subnets_subnet:
                {
                  network_ipam_refs_data_ipam_subnets_subnet_ip_prefix: '172.30.100.0',
                  network_ipam_refs_data_ipam_subnets_subnet_ip_prefix_len: '24',
                },
          }]
     }]
```

MULTIPLE PORTS, ONE MAC

- Child VMI is where the magic happens
- We reference another VMI, the parent
- We set the MAC to parent MAC VMI address
- We assign the vlan

```
vmi_1:
   type: OS::ContrailV2::VirtualMachineInterface
   properties:
     name: 'vmi_1'
     virtual_network_refs: [{ get_resource: child_vn_1 }]
     virtual_machine_interface_properties:
       {
         virtual_machine_interface_properties_sub_interface_vlan_tag: '100',
       }
     virtual_machine_interface_mac_addresses:
       {
         virtual_machine_interface_mac_addresses_mac_address: [{ get_attr: [vm_1, addresses, get_param: 'parent', 0,
"OS-EXT-IPS-MAC:mac_addr"] }],
       }
     virtual_machine_interface_refs: [{ get_attr: [vm_1, addresses, get_param: 'parent', 0, "port"] }]
```

HOW DO I GET THE RIGHT MAC?

- We need to obtain the MAC of the parent
- By checking VM attributes
- We extract it from ports attributes
- Only check on the port connected to the parent network

```
 virtual_machine_interface_mac_addresses:
        {
          virtual_machine_interface_mac_addresses_mac_address: [{ get_attr: [vm_1,
addresses, get_param: 'parent', 0, "OS-EXT-IPS-MAC:mac_addr"] }],
        }
```

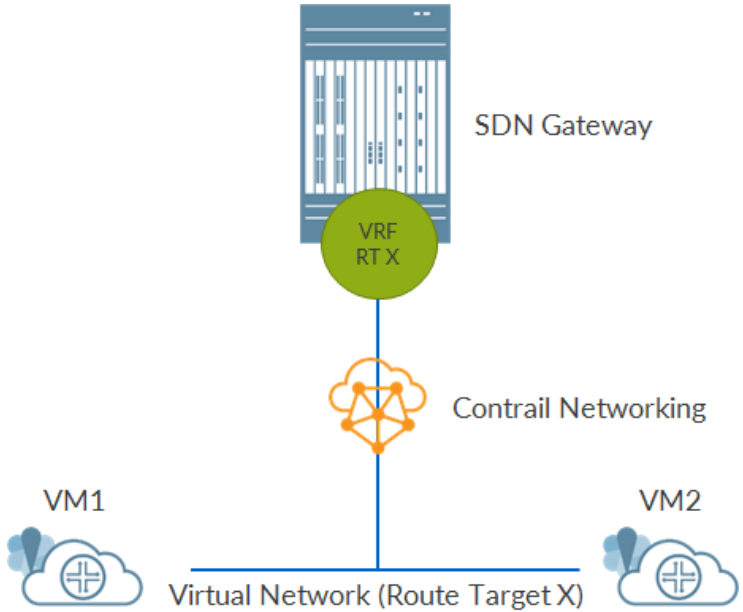
SAME OLD INSTANCE IP

- Still need the Instance IP
- But nothing new here
- Same old Instance IP object


```
iip_1:
   type: OS::ContrailV2::InstanceIp
   depends_on: [ vmi_1 ]
   properties:
     virtual_network_refs: [{ get_resource: child_vn_1 }]
     virtual_machine_interface_refs: [{ get_resource: vmi_1 }]
```

GOING OUT THE DC

- My Virtual Network needs to leave the DC
- In Openstack we have Provider Networks
- Here still tenant networks
- But we use Route targets
- Then same old MP-BGP VPN mechanisms between Contrail and SDN gateway



SDN GATEWAY AS A PE

- Need configuration on SDN gateway as well
- We need a VRF
- Matching route-target
- Usual VRF import/export policies


```
root@sdn-gw# show routing-instances test_vn
instance-type vrf;
interface lo0.1000;
route-distinguisher 192.168.100.1:1;
vrf-import test_imp;
vrf-export test_exp;
vrf-table-label;
```

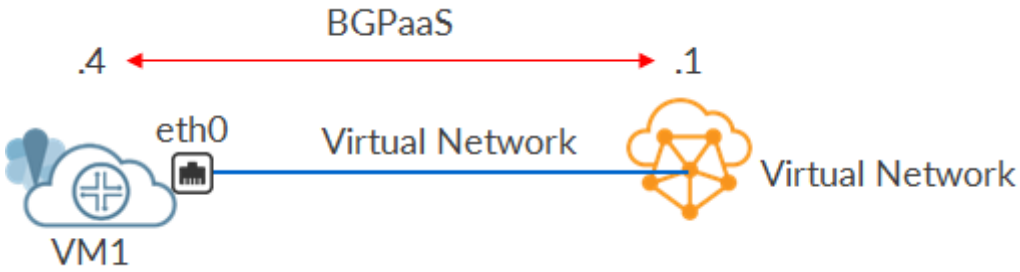
THE RIGHT TARGET

- Virtual Network can have its own route target
- Can import other route-targets
- Can export route-targets

```
resources:
  vip_vn:
    type: OS::ContrailV2::VirtualNetwork
    properties:
      name: { get_param: name}
      import_route_target_list:
        {
          import_route_target_list_route_target: [ 'target:65500:131' , 'target:65500:241' ]
        }
      export_route_target_list:
        {
          export_route_target_list_route_target: [ 'target:65500:201' ]
        }
      route_target_list:
        {
          route_target_list_route_target: ['target:65500:101'],
        }
```

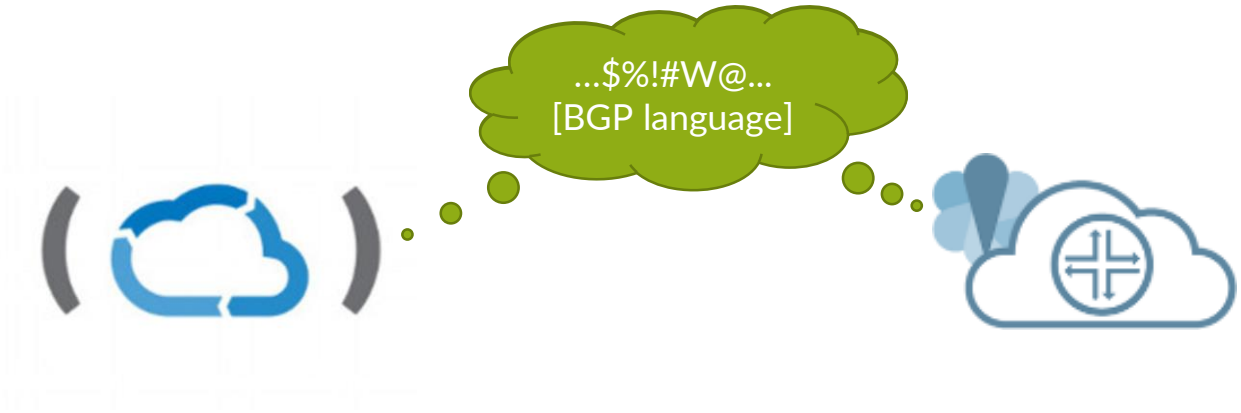
BGP AS A SERVICE

- Totally new feature
- VMs by default are blind to vRouter networking information
- Need to create a communication between vRouter and the VM
- This is BGP as a Service (BGPaaS)



DO YOU TALK BGP?

- VM must talk BGP
- Session with Virtual Network gateway address
- Additional options like prefix limit, as-override, etc…
- BGPaaS object references a VMI



CUSTOMIZE YOUR BGPAAS

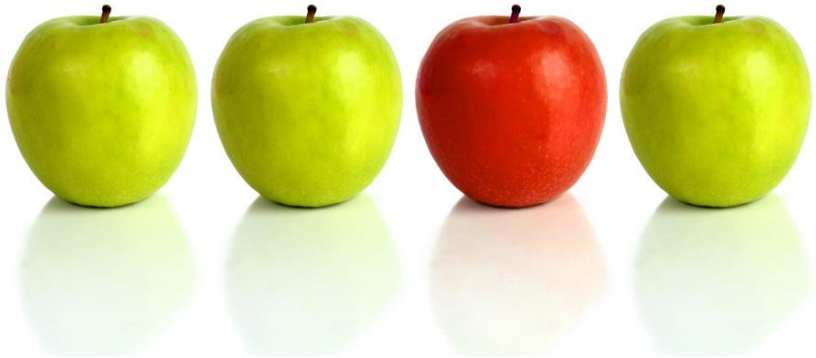
```
bgpaas_2:
   type: OS::ContrailV2::BgpAsAService
   properties:
      name: 'bgp-session-1'
      virtual_machine_interface_refs: [{ get_resource: vmi_2  }]
      autonomous_system: '65511'
      bgpaas_session_attributes:
       {
          bgpaas_session_attributes_address_families:
            {
              bgpaas_session_attributes_address_families_family: ['inet','inet6'],
            },
          bgpaas_session_attributes_family_attributes:
           [{
             bgpaas_session_attributes_family_attributes_address_family: inet,
             bgpaas_session_attributes_family_attributes_loop_count: 1,
             bgpaas_session_attributes_family_attributes_prefix_limit:
               {
                 bgpaas_session_attributes_family_attributes_prefix_limit_maximum: 10
               }
           }]
          bgpaas_session_attributes_as_override: 'true'
          bgpaas_session_attributes_hold_time: '30'
          bgpaas_session_attributes_private_as_action: 'remove'
       }
```

NEUTRON VS CONTRAIL ?

- Neutron objects are available
- We can use them with Contrail
- Not all Contrail objects are compatible with Contrail

WHEN DO WE HAVE TO CHANGE?

- You need additional features that only Contrail can offer
- For example, ECMP fields
- You need to implement something that, conceptually, is totally different between Contrail and vanilla Neutron
- For example, VN that need to go outside the DC



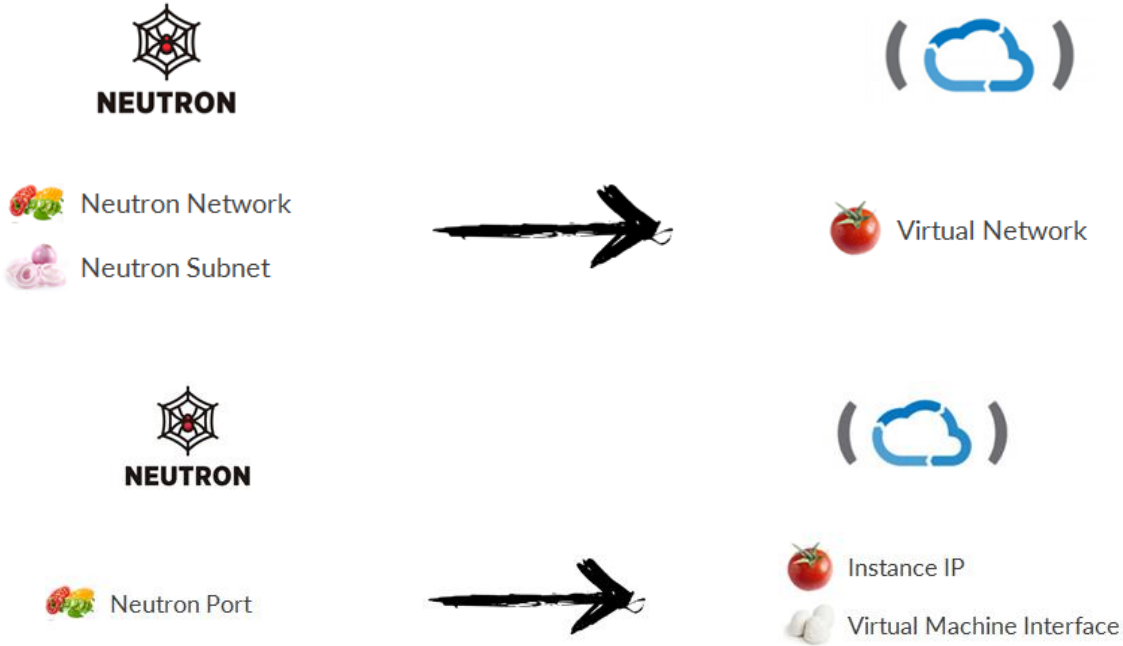
LOST IN TRANSLATION

- We start from a Neutron based template
- And need to move to a Contrail based one
- We tried to identify some common use-cases
- And give best practices on how to transform a stack of Neutron objects in one made with Contrail objects

NEUTRON

CREATING SERVICES IS LIKE A PUZZLE!

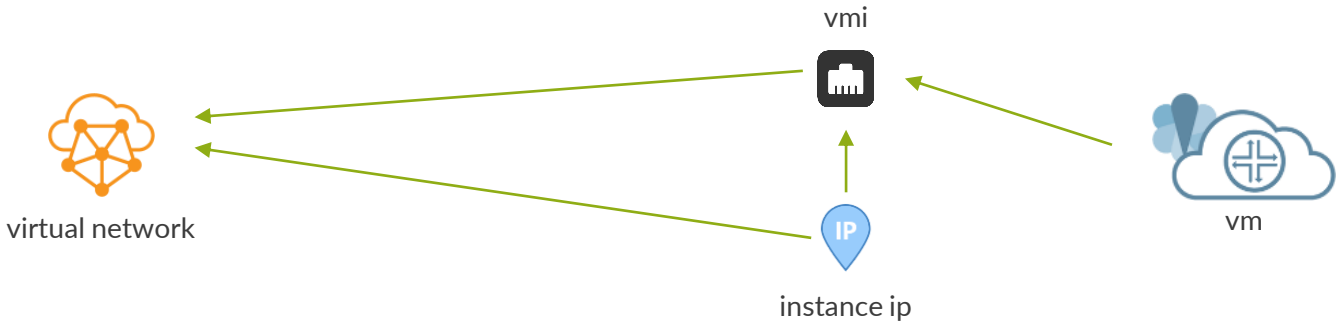
BASIC TRANSLATION



TRANSLATION #1

- Create a VM with one vNIC connected to a new virtual network

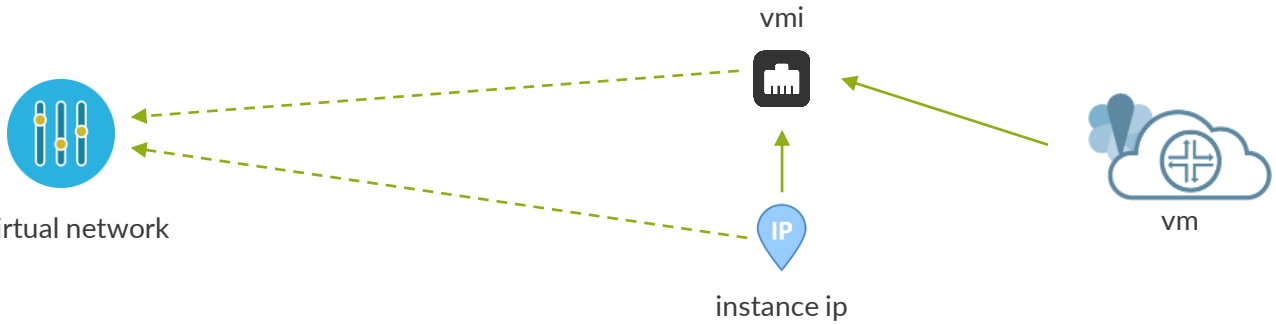
HEAT MAP #1



TRANSLATION #2

- Create a VM with one vNIC connected to an existing virtual network

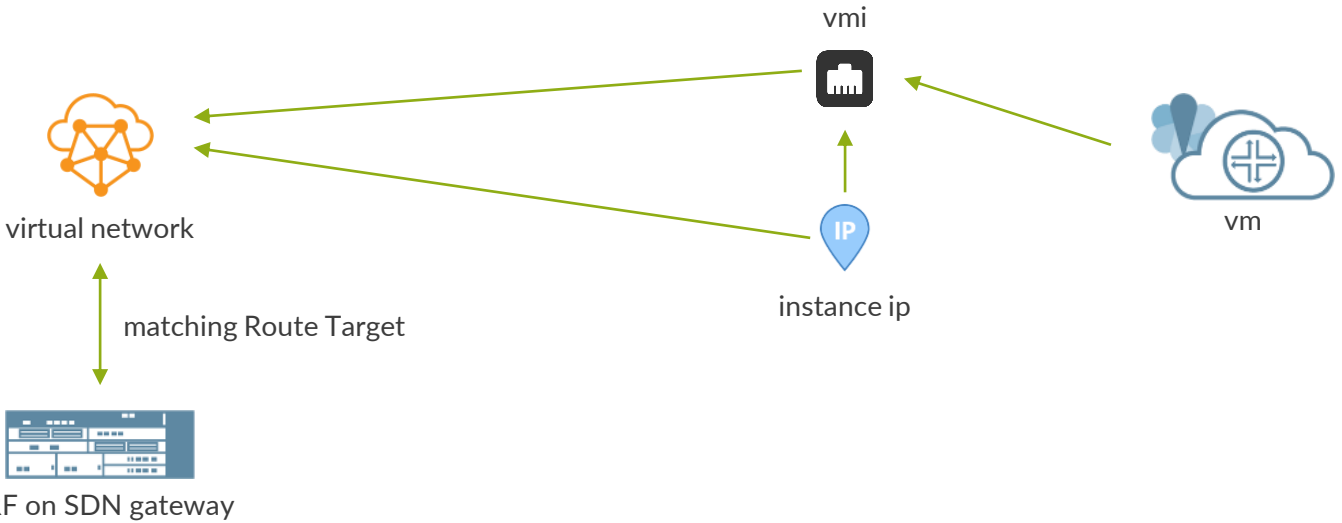
HEAT MAP #2



TRANSLATION #3

- Create a VM with one vNIC connected to a new virtual network that needs to go outside the DC

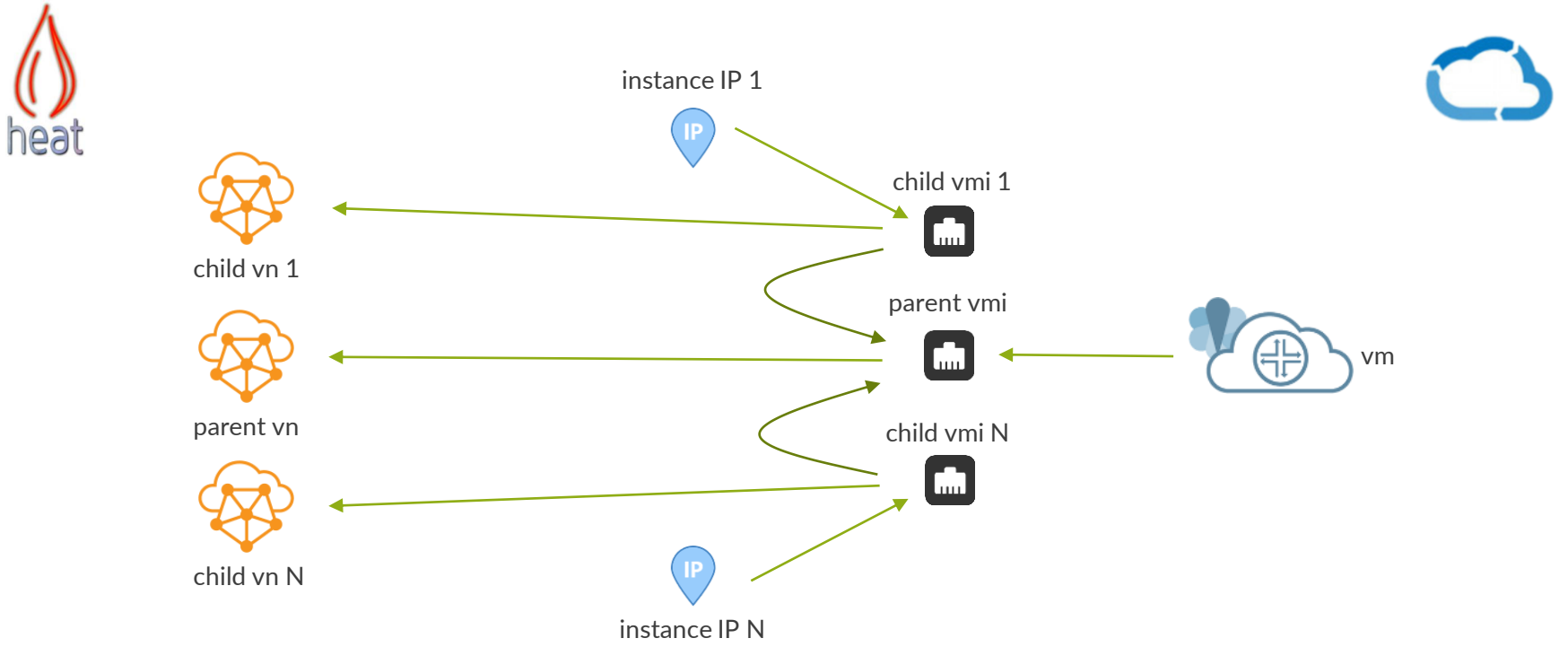
HEAT MAP #3



TRANSLATION #4

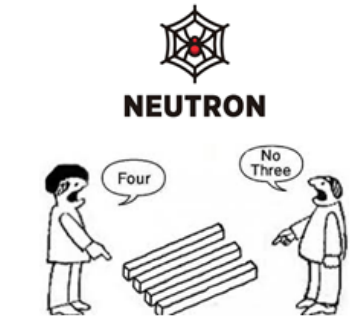
- The VM vNIC needs to be configured as trunk with N vlans

HEAT MAP #5



TRANSLATION #5

- The VM needs to be connected to a L2 network, VXLAN ready

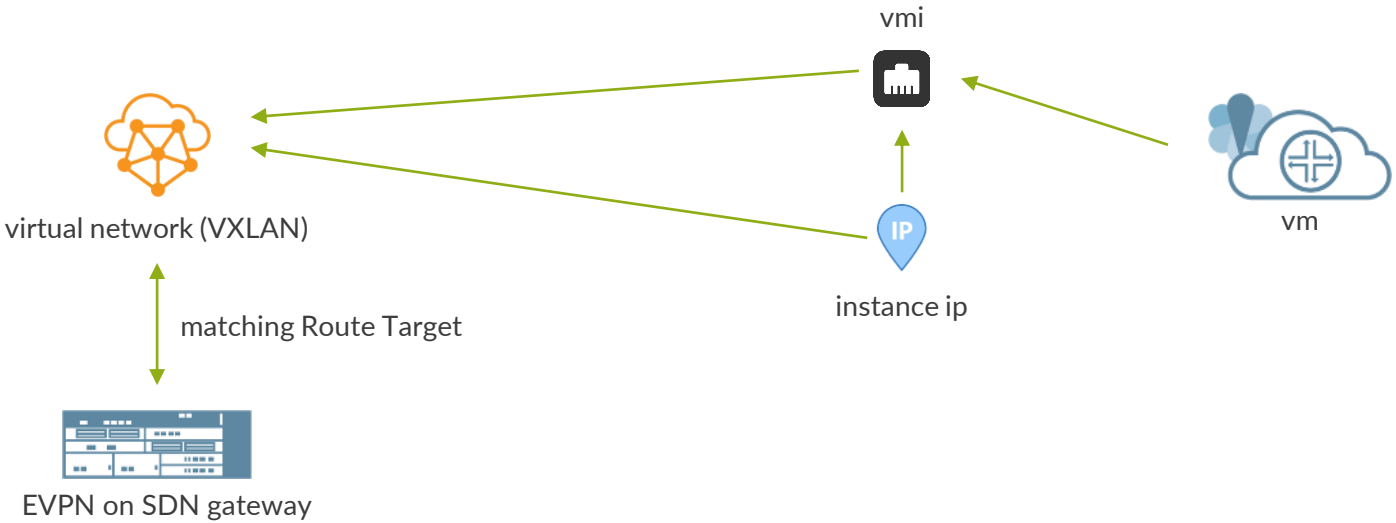


Different concept. Provider network to create a VLAN
Then you need somewhere somehow to include that VLAN into an EVPN

Nova Server

Virtual Network with L2 forwarding mode & user-defined VXLAN id

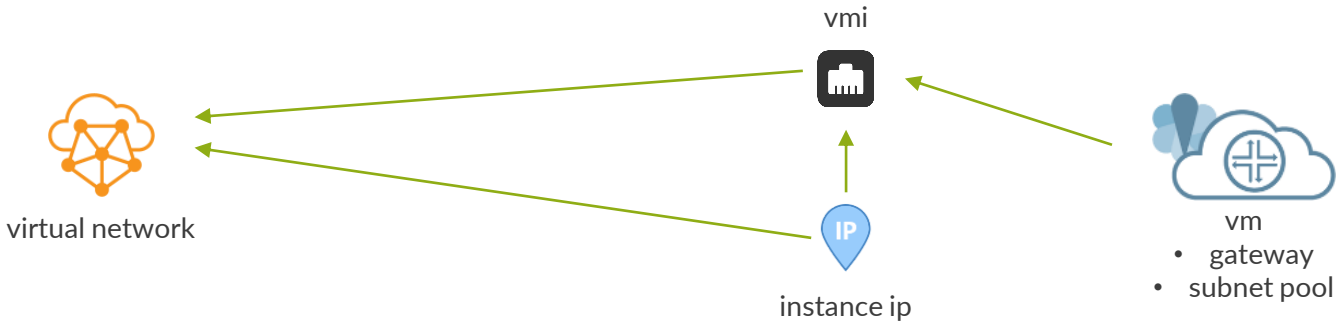
HEAT MAP #5



TRANSLATION #6

- The virtual network needs to use a custom gw and a custom allocation pool

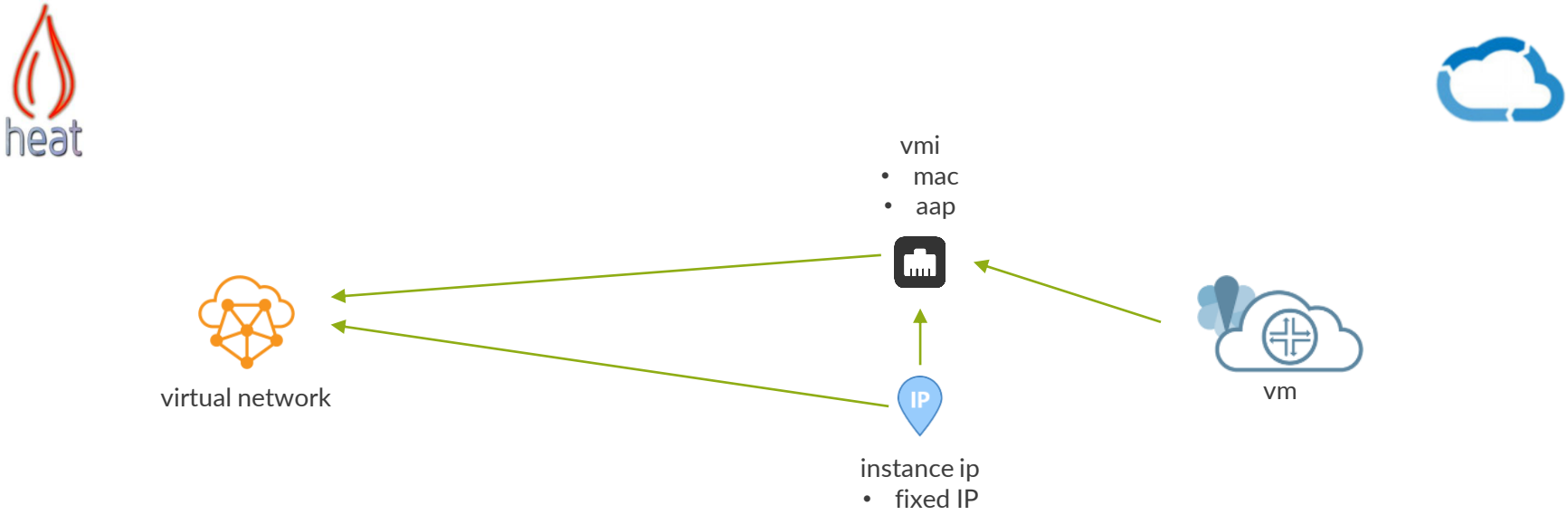
HEAT MAP #6



TRANSLATION #7

- The VM must have a vNIC with a custom MAC, a fixed IP and must allow the configuration of a VIP address

HEAT MAP #7



TRANSLATION #8

- The VM must talk BGP in order to exchange BGP routers and use BFD for faster failure detection

Not Available

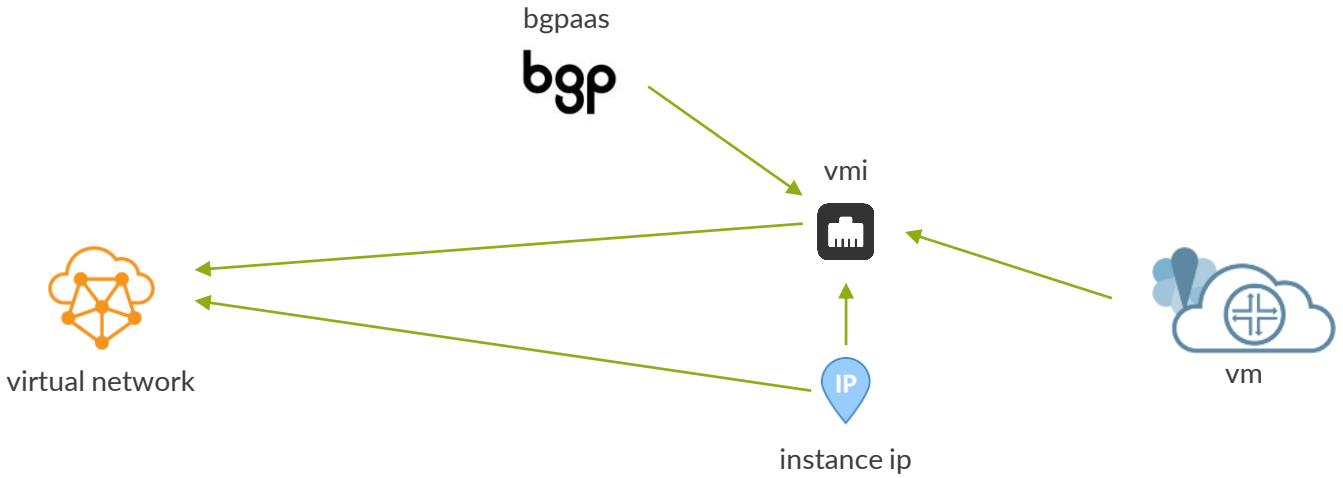
Virtual Network

Virtual Machine Interface

Instance IP

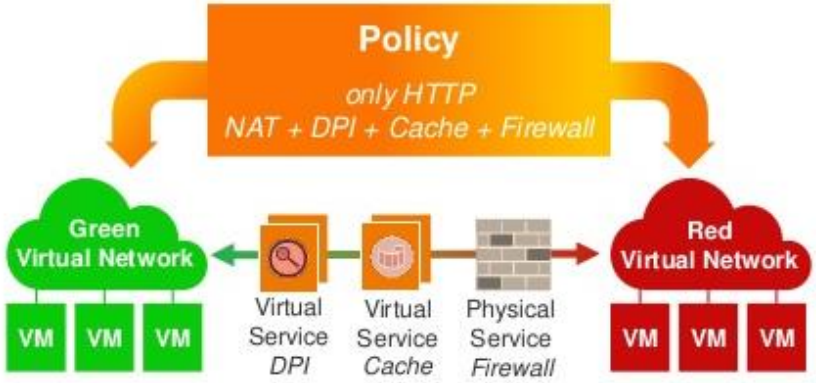
BGPaaS object

HEAT MAP #8



SERVICE CHAINING

- Contrail exclusive feature
- Manage communications between virtual networks
- By adding services between them!

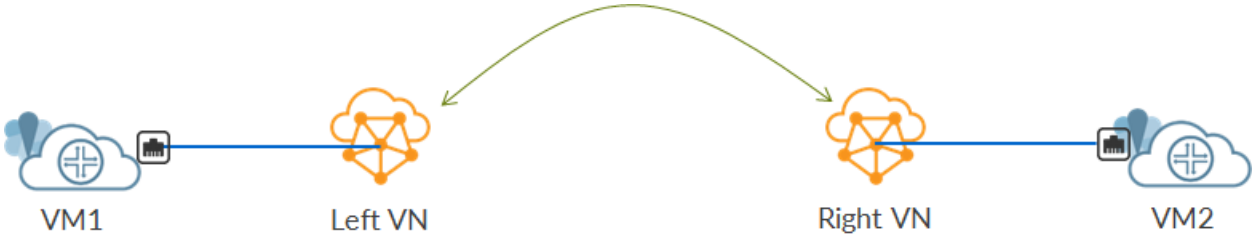


AT FIRST IT WAS TWO NETWORKS

- By default two virtual networks cannot talk to each other
- They are separate broadcast domain
- In a physical network we would need a router

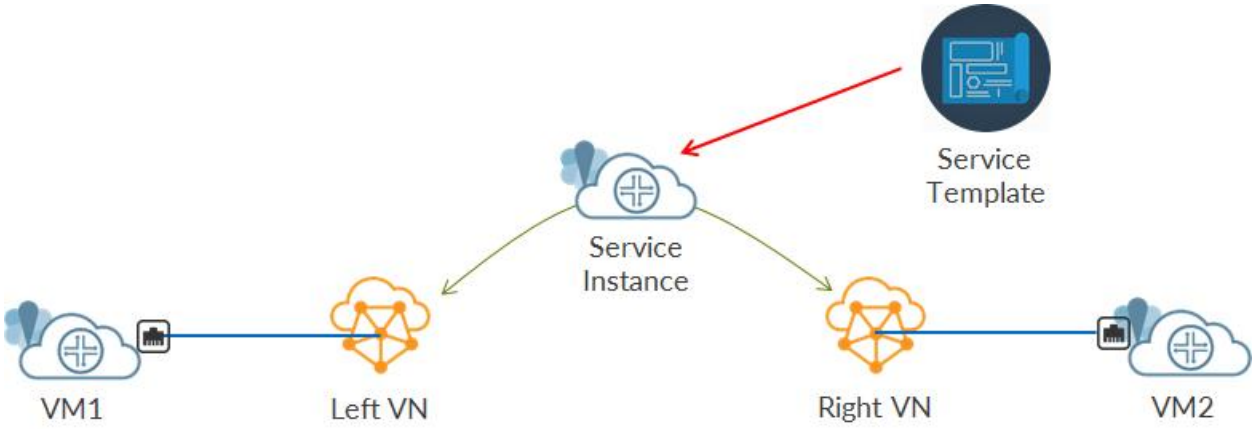
THEN A POLICY CONNECTED THEM

- Networks can talk to each other through a network policy
- Applied to both virtual networks
- A list of rules telling which traffic can pass or not
- Rules built over protocol, dst/src addresses, dst/src virtual networks, dst/src ports



AND A SERVICE CONTROLLED COMMUNICATIONS

- If policy allows traffic to pass we can add one or more service between the two virtual networks
- Service can be a firewall
- We bring security between two virtual networks
- Service chaining routing is automatically managed by Contrail

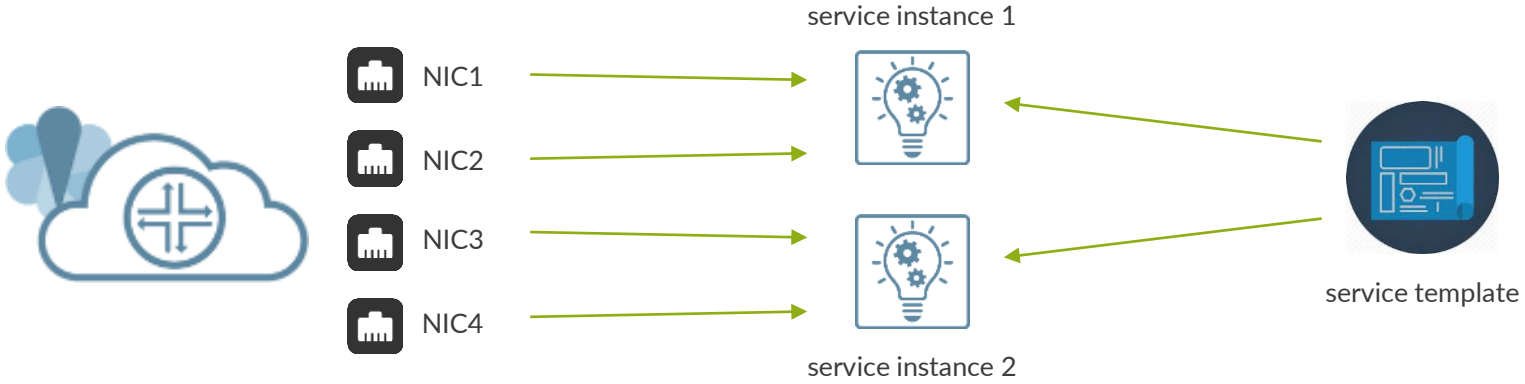


SERVICE CHAIN RECIPE

- Two virtual networks
- One network policy applied to those networks
- One or more VMs acting as service elements
- Service templates describing the nature of the service instance
- Port tuple, a collection of ports belonging to a VM
- Service instance, the actual service part of the service chain

DON'T BE CONFUSED!

- Based on the template
- Uses ports of a VM as defined in a port tuple
- A service instance is not a VM!
- We might have multiple service instances using the same VM



SERVICE CHAIN BUILDING BLOCKS

- Network policy
- Service template
- Service instance
- Port tuple
- Modified VMI
- Service VM

WRITING YOUR POLICY

```
type: OS::Contrailv2::NetworkPolicy
   properties:
     name: { get_param: pol_name }
     network_policy_entries: {
         network_policy_entries_policy_rule: [{
           network_policy_entries_policy_rule_direction: { get_param: direction },
           network_policy_entries_policy_rule_protocol: { get_param: protocol },
           network_policy_entries_policy_rule_src_ports: [{
               network_policy_entries_policy_rule_src_ports_start_port: { get_param: src_port_start },
               network_policy_entries_policy_rule_src_ports_end_port: { get_param: src_port_end }
             }],
           network_policy_entries_policy_rule_dst_ports: [{
               network_policy_entries_policy_rule_dst_ports_start_port: { get_param: dst_port_start },
               network_policy_entries_policy_rule_dst_ports_end_port: { get_param: dst_port_end }
             }],
           network_policy_entries_policy_rule_dst_addresses: [{
               network_policy_entries_policy_rule_dst_addresses_virtual_network: { get_param: dst_add }
             }],
           network_policy_entries_policy_rule_src_addresses: [{
               network_policy_entries_policy_rule_src_addresses_subnet: {
                   network_policy_entries_policy_rule_src_addresses_subnet_ip_prefix: {get_param: rule_cidr_1},
                   network_policy_entries_policy_rule_src_addresses_subnet_ip_prefix_len: {get_param: rule_mask_1},
                 }
             }],
           network_policy_entries_policy_rule_action_list: {
               network_policy_entries_policy_rule_action_list_simple_action: { get_param: simple_action },
               network_policy_entries_policy_rule_action_list_apply_service:  [{ get_param: svc_inst_fq_name }]
             },
         }]
       }
```

APPLYING POLICY TO A VIRTUAL NETWORK

- Policy must be assigned to virtual networks too
- To both virtual networks, left and right

```
left_vn:
   type: OS::ContrailV2::VirtualNetwork
   depends_on: [ net_policy_apn_1 ]
   properties:
     ...
     network_policy_refs: [{ get_param: pol_fq_name }]
     network_policy_refs_data:
       [{
         network_policy_refs_data_sequence:
           {
             network_policy_refs_data_sequence_major: 0,
             network_policy_refs_data_sequence_minor: 0,
           },
       }]
```

DRAW YOUR TEMPLATE

- Defines service characteristics
- Type, mode, interfaces

```
svc_tmplt:
   type: OS::ContrailV2::ServiceTemplate
   properties:
     name: { get_param: service_template_name }
     service_template_properties:
       {
         service_template_properties_version: '2',
         service_template_properties_service_mode: 'in-network',
         service_template_properties_service_type: 'firewall',
         service_template_properties_interface_type:
           [
             {
             service_template_properties_interface_type_service_interface_type: 'left',
             },
             {
             service_template_properties_interface_type_service_interface_type: 'right',
             }
           ],
         service_template_properties_ordered_interfaces: 'true',
       }
     domain: 'default-domain'
```

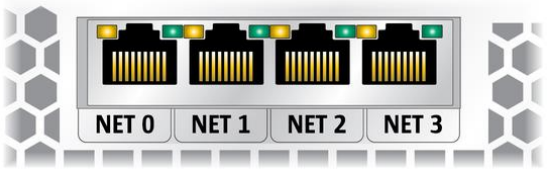
DEFINE THE SERVICE INSTANCE

- We bind interfaces to virtual networks

```
svc_inst_apn_1:
   type: OS::ContrailV2::ServiceInstance
   depends_on: [ svc_tmplt ]
   properties:
     name: { get_param: svc_inst_name }
     service_instance_properties:
       {
         service_instance_properties_interface_list:
           [
             {
             service_instance_properties_interface_list_virtual_network:
               {
                  list_join: [':', { get_attr: [ left_vn, fq_name ] } ]
               },
             },
             {
             service_instance_properties_interface_list_virtual_network:
               {
                  list_join: [':', { get_attr: [ right_vn, fq_name ] } ]
               },
             }
           ],
       }
     service_template_refs: [{ get_resource: svc_tmplt }]
```

GATHER PORTS INTO A TUPLE

- Binds a list of interfaces
- References a service instance



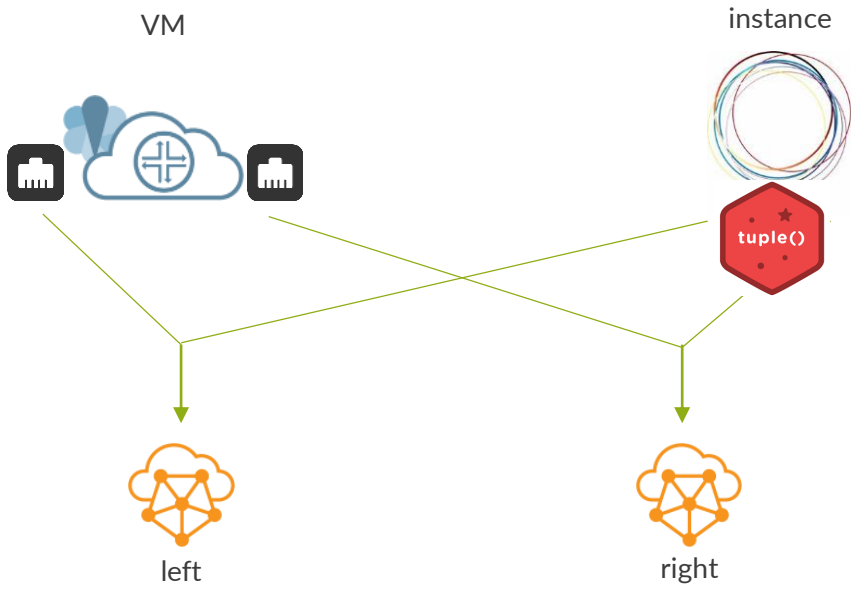

```
prt_tpl_1:
   type: OS::ContrailV2::PortTuple
   depends_on: [ svc_inst_apn_1 ]
   properties:
     name: { get_param: tuple_name }
     service_instance: { list_join: [':', { get_attr: [ svc_inst_apn_1, fq_name ] } ] }

vmi_2:
   type: OS::ContrailV2::VirtualMachineInterface
   depends_on: [ prt_tpl_1 ]
   properties:
     virtual_machine_interface_properties:
       {
         virtual_machine_interface_properties_service_interface_type: 'left',
       }
     port_tuple_refs: [{ get_resource: prt_tpl_1 }]
     virtual_network_refs: [{ list_join: [':', { get_attr: [ left_vn, fq_name ] } ] }]
```

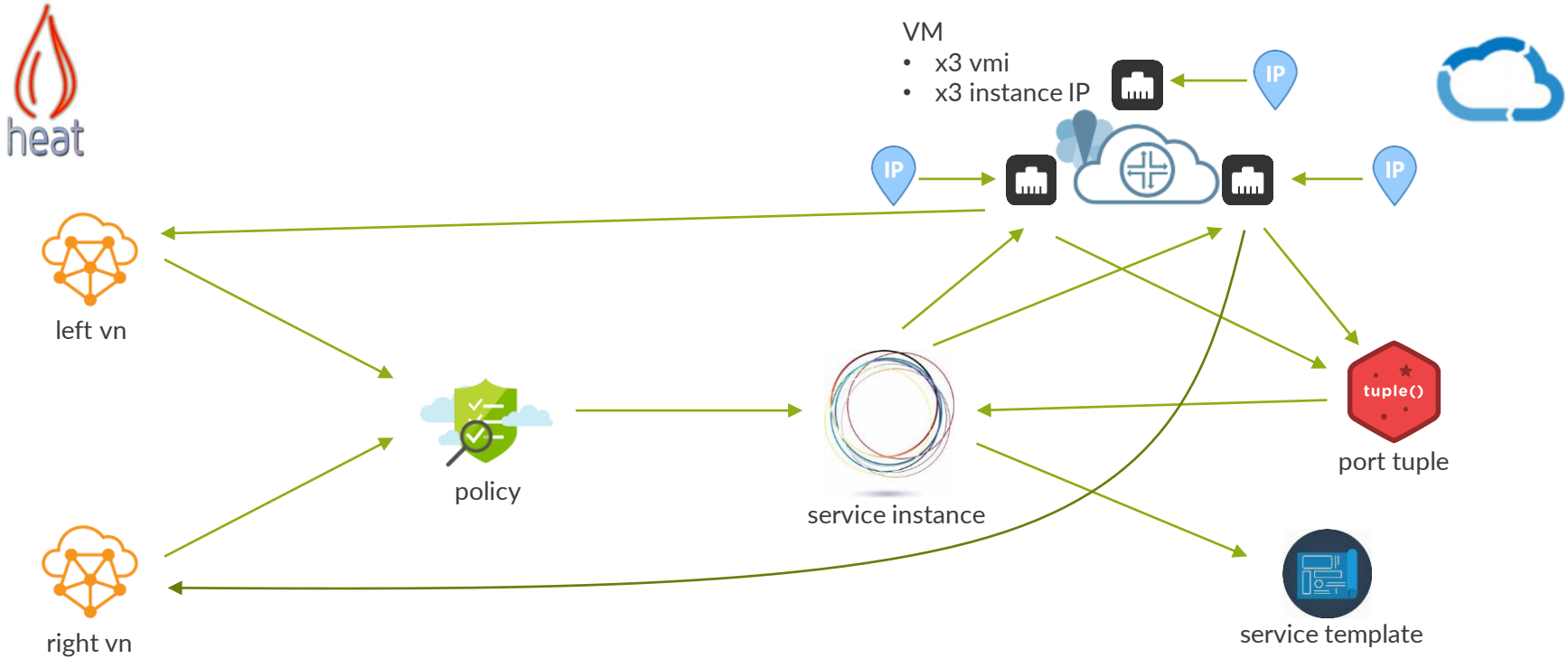
SERVICE VM WILL MANAGE TRAFFIC

- Standard VM
- Some of its ports assigned to the service instance

```
vm1:
   type: OS::Nova::Server
   depends_on: [ iip_1, iip_2, iip_3 ]
   properties:
     name: {get_param: vm1name }
     image: { get_param:  image }
     flavor: { get_param: flavor }
     networks:
       - port: { get_resource: vmi_1 }
       - port: { get_resource: vmi_2 }
       - port: { get_resource: vmi_3 }
```

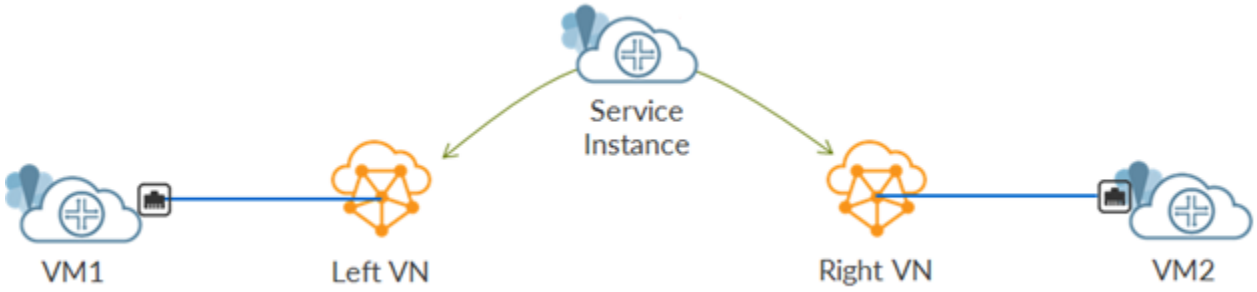


SERVICE CHAINING HEAT MAP



USE CASES? SKY IS THE LIMIT

- Policy allows inter-VN communication
- Service configured between the two VN
- Service Firewall blocks some traffic!



SUMMING UP

- SDN brings automation
- Automation in Openstack means Heat
- Contrail has its own Heat objects
- Models to create VNFs with Contrail
- Best practices to translate Neutron templates to Contrail templates

THAT'S ALL!