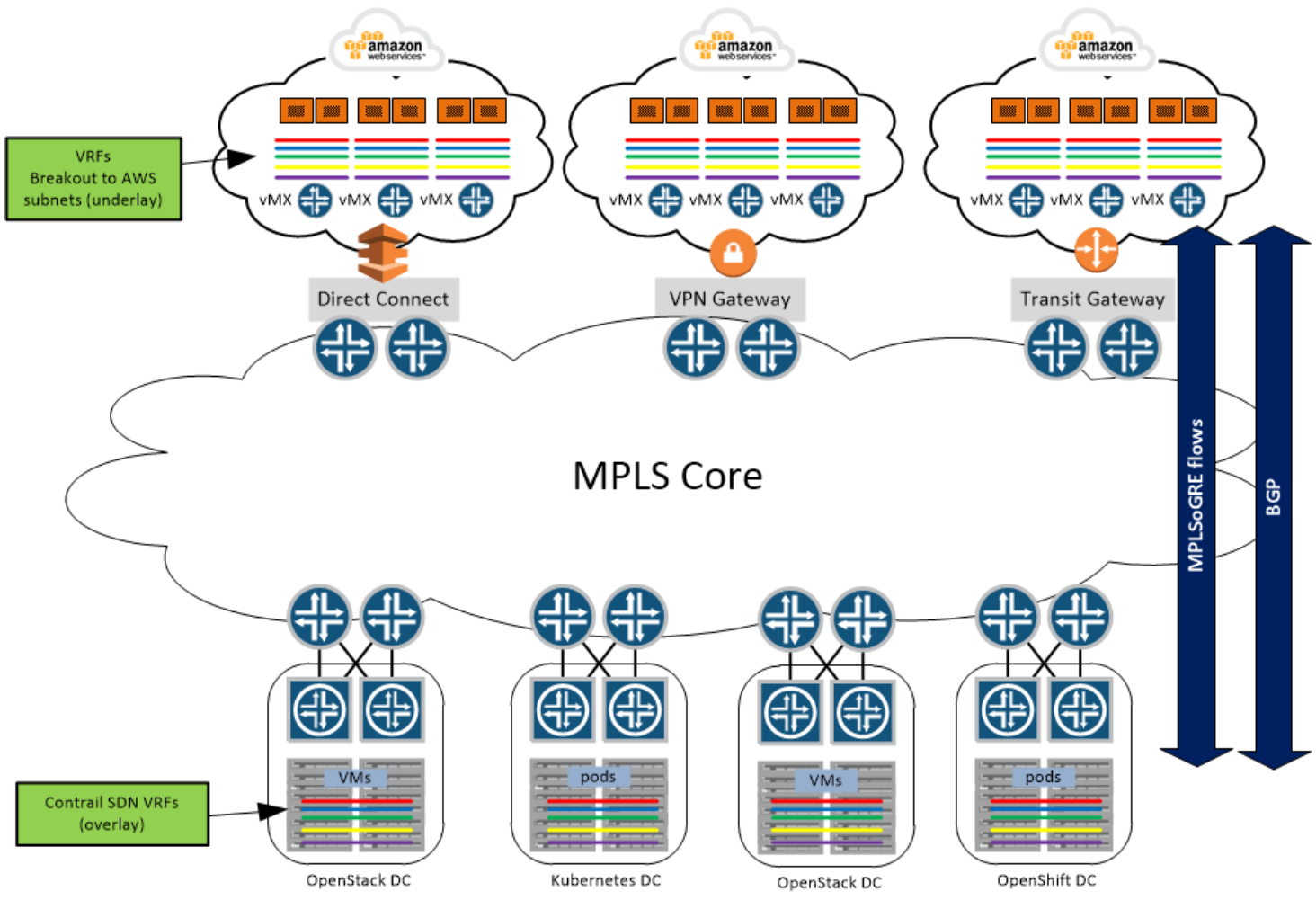
Stretching on premise Datacenter's running contrail SDN out to AWS native EC2 instances using vMX's as SDN gateways

Last edited by **Simon Green** just now

The figure below shows the use case: stretching SDN networks from on premise DC's out to native AWS services.

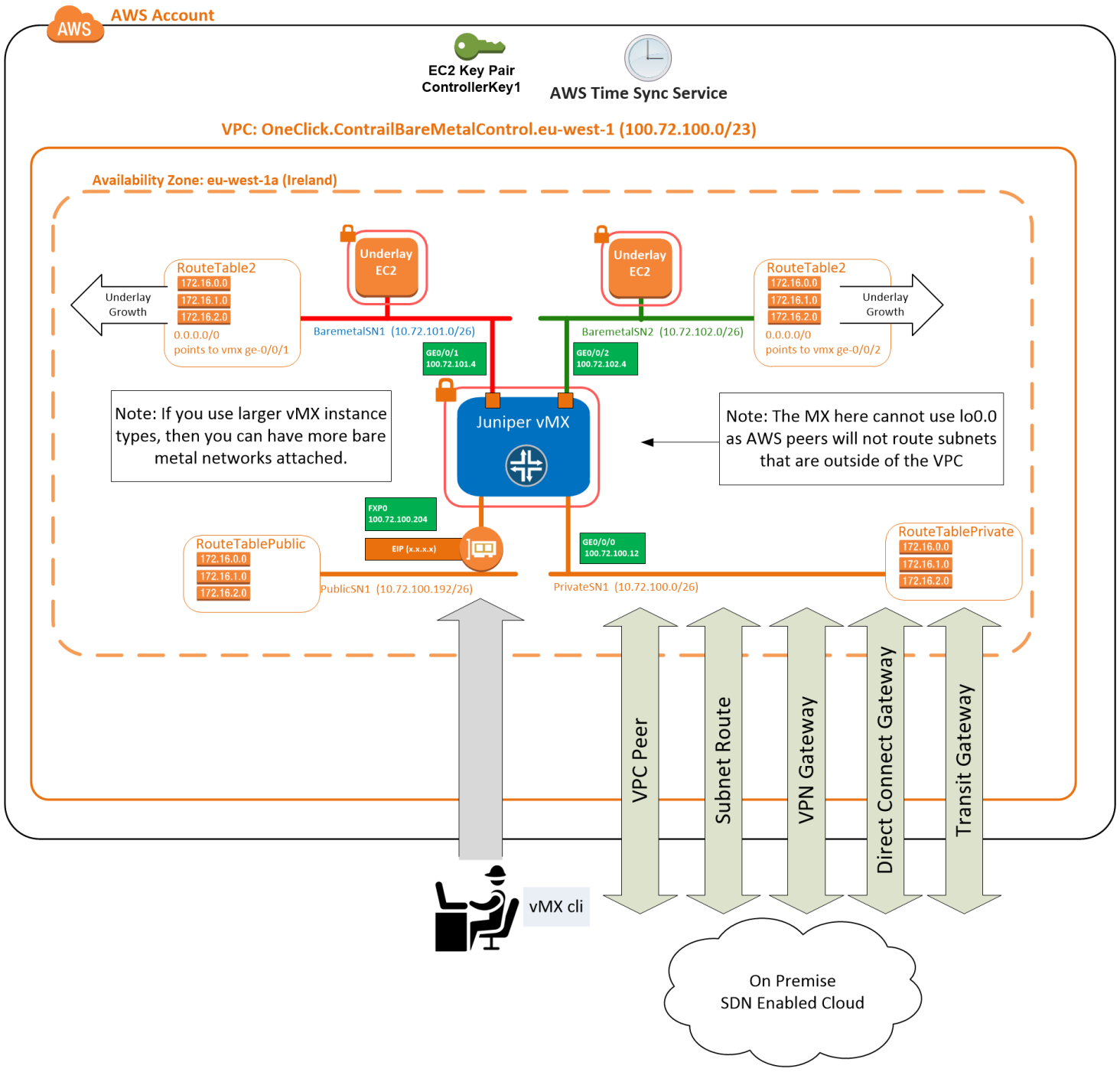


1. Here we have one or more on premise Datacenter's running contrail SDN (OpenStack, Kubernetes, OpenShift, VMWare).
2. Contrail is of course designed to allow us to automagically extend VRF's between the data centres using controller federation and route targets.
3. Here we wish to connect our on premise VMs/Pods to native AWS services, in this case EC2 instances. These native AWS services have noting to do with Contrail SDN. We would like to maintain the same benefits of automated multi-tenancy that we have within the SDN enabled DC's all the way into AWs.
4. So in short we want to be able to automatically extend the SDN networks in our DC's, into AWS and breakout locally to thee VPC's services.

In our previous related WIKI's and videos I used Contrail fabric to configure a vMX based SDN gateway for an OpenStack, also deployed into AWS.
https://svl-ssd-git.juniper.net/sre/aws-one-click-deployers/wikis/Deploy-a-simple-OpenStack-test-setup-into-AWS-with-Contrail-Command
https://svl-ssd-git.juniper.net/sre/aws-one-click-deployers/wikis/Brownfield:-Adding-a-vMX-into-AWS,-then-using-Contrail-Command-Fabric-manager-to-configure-it-as-an-SDN-Gateway

That worked well within AWS. It would also work well within an on premise DC. It would not work well between AWS and an on premise which is our use case here. This is because Fabric manager uses loopback interfaces within the vMX (lo0.0) to terminate the BGP and MLS traffic. AWS however will not pass traffic which is not within it's VPC subnets over a transport such as a peer, VPN, Direct connect (nope sorry you also cannot move lo0.0 into the VPC subnets due to other AWS routing limitations).

So the good news is that here I provide a fully automated stack, no fabric, just click the link and it will build the vMX, configure it as an SDN gateway, then add the supporting AWS resources. All based on the stack parameters you provide it.



So what about the on premise DC to AWS VPC connectivity

AWS have three basic methods to connect your on premise out to a public cloud VPC.

1. VPC native VPN gateways (IPSEc)
2. VPC Direct connect gateways (using a direct connect clan via an IPX)
3. Transit gateways (using a mixture of the above)
4. VPC peers (routing between accounts and other VPCs

I'm using VPC VPN gateways in my laboratory, however all of these will work fine. Contrail only requires ip to ip connectivity between the on premise DCs and the AWS VPC (BGP, MPLSoGRE), it has no opinion on how you provide it.

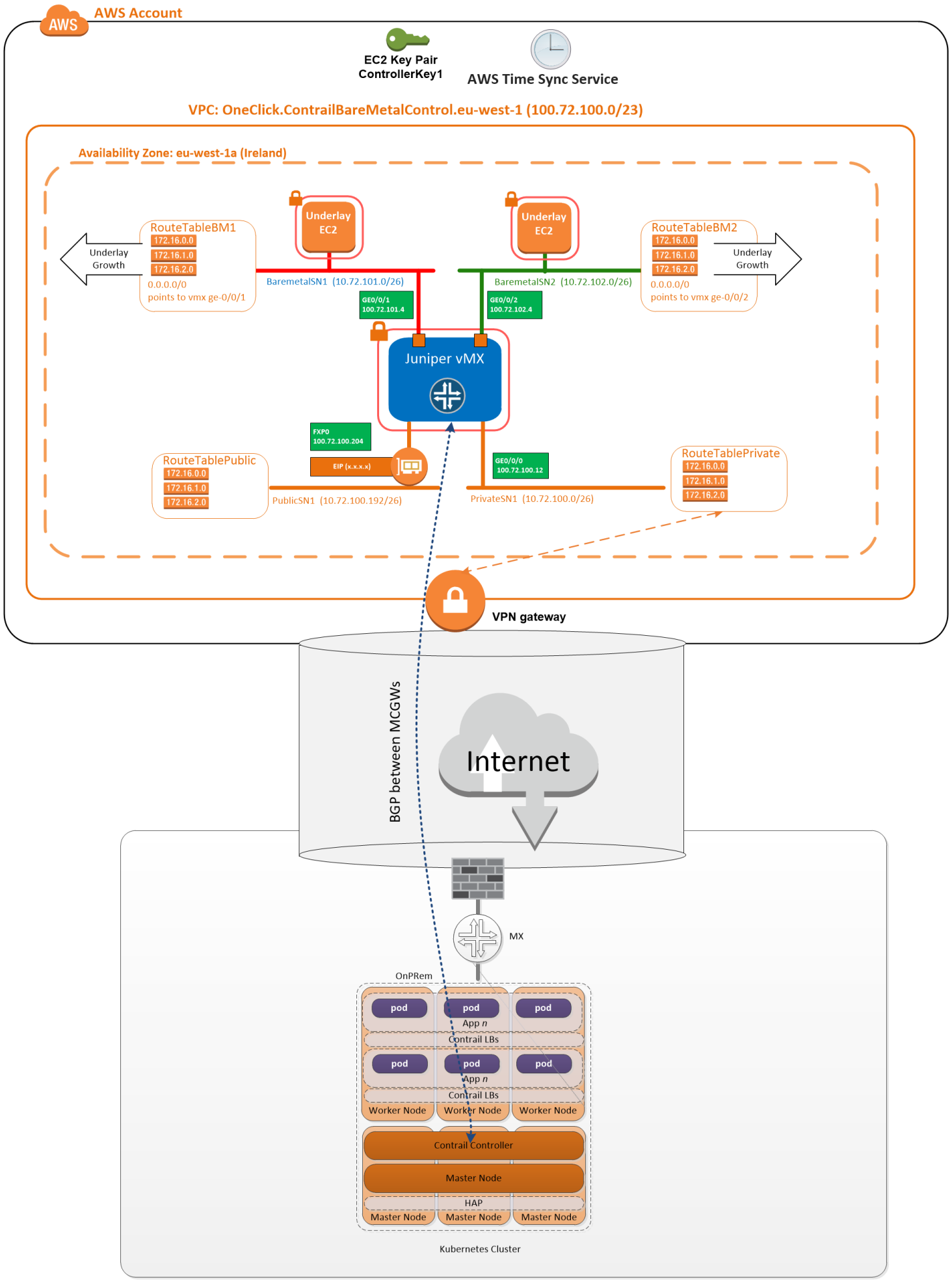
In this wiki I am not going to show how I bring up a VPN gateway and connect it to an IPSec GW (Juniper SRX in my case) as the AWS documentation is great for those details.

In this wiki I'm also not covering how to build the on premise DCs. The assumption here is you already have Contrail SDN running and integrated into some cloud orchestrators on premise.

I am going to focus on the AWS side of things. Then explain how to peer your on premise private cloud DC into your AWS pubic cloud VPC in an automated way.

Here is my laboratory setup

*You can see I have an on premise Kubernetes cluster (it could be any type of Contrail enabled DC; OpenStack, VMWare, Kubernetes, OpenShift). *I'm using an SRX to handle the IPSEc out to the AWS VPC gateway (could be any type of AWS gateway or peer). *In AWS itself I've deployed my stack which builds a vMX configured as an SDN gateway with underlay breakout, public JunOS cli connectivity (ssh). A private subnet for BGP and MPLSoGRE. Some bare metal networks and test bare metal EC2 instances, so we can see it work.



deployment steps Lets deploy the AWS stack.

One Time Setup:

1. You need an AWS account. Login to the AWS console with your browser.
2. In your AWS account console you need to an EC2 SSH key pair named "ContrailKey" (unless you change the name in the stack parameters Note: You should be able to deploy this stack in most regions within AWS, I'm using London here.

I have two stack options. a. The Greenfield deploys the AWS VPC and all of the services within it. b. The brownfield deploys all of the services into your existing AWS VPC. The reason for two stack options is that AWS VPC gateways (IPSEC VPN gateways for example) get deleted every time you delete a VPC, and that can be a real pain. So deploy the Greenfield stack, add your VPN gateway, delete the stack and your VPN gateway gets deleted as well. With the brownfield stack you keep the VPC and the gateway that you. deployed before hand.

Deploy the vMX CloudFormation stack as a BrownField (The VPC already exists)

1. Click this link to launch the CloudFormation stack into your account (UK) Launch Stack

Deploy the vMX CloudFormation stack as a Greenfield (I create the VPC)

1. Click this link to launch the CloudFormation stack into your account (UK) 

Note: if you right click either stack and copy the link, then share it with your customers via email. It will work for them as well

---

->NEXT
complete the field idVPC (Only required for the brownfield stack)
complete the field ContrailbgppeerIP1 This is your first Contrail controller node IP x.x.x.x
complete the field ContrailbgppeerIP2 This is your first Contrail controller node IP x.x.x.x. If you are not using HA Make it the same as node1
complete the field ContrailbgppeerIP3 This is your first Contrail controller node IP x.x.x.x. . If you are not using HA Make it the same as node1
->NEXT
->NEXT
->Deploy Stack Note: all fields can be changed, but I recommend that first time you go with the default.

Here are some other fields to be aware of, that you might want to change from the defaults for your setup.

```
**ContrailUnderlayNetwork** This is a subnet covering the Contrail controllers and compute
**ContrailOverlayNetworks** This is a subnet covering contrails overlay networks
**ContrailbgpASN** This is the ASN of your contrail cluster
**KeyName** This is the SSH key for the vMX (user jnpr) and the bare metal ec2 instances (u
**SGSubnet1,2,3,4** These are additional DC subnets ythat ou would like to reach the vMX
**UserLocation** This is the subnet you would like to allow for vMX clil access, your lapto
**BareMetal1RouteTarget** This is the route target for stretching Bare metal network 1. Def
**BareMetal2RouteTarget** This is the route target for stretching Bare metal network 2. Def
```

If you prefer to use AWS cli to deploy the stack, here is a Greenfield stack example

```
aws cloudformation create-stack \
--stack-name SDN-GW-Brownfield-Automated \
--disable-rollback \
--template-url https://s3-eu-central-1.amazonaws.com/contrail-one-click-deployers/vMX-SDN-G
--parameters \
ParameterKey=idVPC,ParameterValue="vpc-0ea16152c63b68355"
```

Here is a greenfield example

```
aws cloudformation create-stack \
--stack-name SDN-GW-Greenfield-Automated \
--disable-rollback \
--template-url https://s3-eu-central-1.amazonaws.com/contrail-one-click-deployers/vMX-SDN-G
```

The stack will complete in a few minutes.

A small and light VMX, which is the default here, will then take another 10-15 minutes to come up fully. The EC2->Instances view will show check 2 of 2 passed once it ready.

as ever the output of the CloudFormation stack will show you how to connect to the vMX cli. it. will also tell you the bgp ip address to peer into on contrail.

---

add aws config to allow your on prem to connect to the aws subnets

on the AWS console enable bgp route propagation between the private subnet and your VPC gateway
console->vpc->route tables->ControllerPrivateRouteTable1.SDNGateway1.eu-west-2a
->Route_Propagation
enable

you should see routes (at least one) propagated from your on premise gateway
for example:

```
172.30.188.0/24 vgw-0119dabd25861b92d   active Yes
192.190.0.50/32 vgw-0119dabd25861b92d   active Yes
192.190.0.55/32 vgw-0119dabd25861b92d   active Yes
```

Adding the BGP peer into your on premise DC's contrail controller

contrail ui.
config->infrastructure->bgp routers->
Router Type=BGP Router
Host Name=ip-100-72-100-204
Vendor ID=Juniper
IP Address=100.72.100.12
Router ID=100.72.100.12
AS=64512
AddressFamilies=inet, e-vpn, inet-vpn, route-target
BGP Peers:
Associate Peers:
controller1
controller2 (if HA)
controller3 (if HA)

Check the status:
Monitor->Control Nodes->Controller1->Peers
100.72.100.12 Established xx/xx

Check the status on the vMX

```
show bgp summary
172.30.188.13         64512         16         36       0       0          39 Establ
bgp.evpn.0: 6/6/6/0
bgp.l3vpn.0: 2/2/2/0
bgp.rtarget.0: 5/7/7/0
172.30.188.14         64512         39         19       0       0          39 Establ
bgp.evpn.0: 4/6/6/0
bgp.l3vpn.0: 3/3/3/0
bgp.rtarget.0: 7/11/11/0
172.30.188.15         64512         40         42       0       0          39 Establ
bgp.evpn.0: 1/7/7/0
bgp.l3vpn.0: 3/3/3/0
bgp.rtarget.0: 7/11/11/0
```

Adding the fabric subnet to the contrail controller

contrail ui.
config->global config->ip fabric subnets
->100.72.100.0/26 #this is the aws vmx private subnet

checking the overlay tunnel encapsulation priority order

contrail ui.
config->global config->encapsulation order priority
You need to have MPLSoGRE somewhere in the list, so it's supported.

stretching an on premise SDN network default pod networks out to the vMX bare metal 1 network

contrail ui
->select the k8sdefault project
->config->networks->k8s-default-pod-network
->route targets
64512 1001

stretching an on premise SDN network, isolated namespace out to the vMX bare metal 2 network

contrail ui
->select the deposits project
->config->networks->k8s-default-pod-network
->route targets
64512 1002

testing connectivity DC default pod to bare metal native EC2 in AWS

on Kubernetes

```
get pods -o  wide
NAME                        READY   STATUS    RESTARTS   AGE     IP              NODE
dc1-pod-57fc6c9bf9-wc68w   1/1     Running   0          6m42s   10.47.255.250   dc-compute1

kubectl exec -i -t dc1-pod-57fc6c9bf9-wc68w /bin/bash
ping 100.72.101.33
PING 100.72.101.33 (100.72.101.33) 56(84) bytes of data.
64 bytes from 100.72.101.33: icmp_seq=1 ttl=63 time=13.4 ms
```

so from a pod on the DC in a contrail overlay network
I can ping out to a bare metal ec2 sat in an AWS ec2, behind the MX router.

testing connectivity isolated namespace pod to EC2 bare metal in aws

on Kubernetes

```
get pods -o wide --namespace=deposits
NAME                                READY   STATUS    RESTARTS   AGE    IP              NO
deposits-celery-59fd6fcf6f-jnbvm    1/1     Running   0          157m   10.47.255.246   dc
deposits-python-b94b856bc-zt4xc     1/1     Running   0          157m   10.47.255.245   dc
deposits-rabbitmq-75769cbbc7-h2fmp  1/1     Running   0          157m   10.47.255.249   dc
deposits-redis-f49dc6cb7-httk4      1/1     Running   0          157m   10.47.255.248   dc
kubectl exec -it deposits-celery-59fd6fcf6f-jnbvm --namespace=deposits /bin/bash
ping 100.72.101.108
PING 100.72.101.108 (100.72.101.108) 56(84) bytes of data.
64 bytes from 100.72.101.108: icmp_seq=1 ttl=63 time=12.8 ms
64 bytes from 100.72.101.108: icmp_seq=2 ttl=63 time=11.7 ms
```

testing bare metal native EC2 in AWS to dc

#from the vMX

```
start shell
[put the aws ec2 ssh key into file key.txt]
chmod 0400 key.txt
ssh -i key.txt ubuntu@100.72.101.12
#ping the pod in the DC
ping 10.47.255.250
PING 10.47.255.250 (10.47.255.250) 56(84) bytes of data.
64 bytes from 10.47.255.250: icmp_seq=1 ttl=62 time=33.7 ms
64 bytes from 10.47.255.250: icmp_seq=2 ttl=62 time=11.9 ms
```

tracing the DC SDN flows, between the DC and AWS

on the pod compute (compute1)

```
tcpdump -i ens192 proto gre
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens192, link-type EN10MB (Ethernet), capture size 262144 bytes
00:53:29.245391 IP dc-compute1 > 100.72.100.12: GREv0, length 92: MPLS (label 16, exp 0, [S
00:53:29.257034 IP 100.72.100.12 > dc-compute1: GREv0, length 92: MPLS (label 23, exp 0, [S
```

Scaling up

- The intention is that you deploy multiple stacks either into one VPC or Multiple VPCs in order to scale up. With a small edit these stacks will be able to share public and private subnets.
- Additionally larger vMX instance types will allow more ports (for example an m4.4xlarge allows 8) you can then add additional ports and subnets to the stack.
- Larger vMX instance types and you can change the vMX config from light to performance mode, much faster.
- Each bare metal subnet has its own AWS router. If you attach additional subnets to the router, then add a route to the vMX VRF config they will all be reachable from Contrail.

Note: Although my laboratory uses Kubernetes on premise, Contrail SDN also integrates in DC's running OpenStack, OpenShift-Brownfield, VMware. They will also work precisely the same way.