Cross-Cloud Connectivity

Diego Casati – Senior Software Engineer
Commercial Software Engineering
DCASATI(1) BSD General Commands Manual DCASATI(1)

NAME
dcasati -- Diego Casati: a (curious) Engineer

SYNOPSIS
dcasati [--network-engineering] [--software-define-(network|storage)]
         [--software-development] [--orchestration] [--devops]

HISTORY

Diego Casati is a Senior Software Engineer for Microsoft focusing on
Kubernetes, Networking and Linux and BSDs. Prior to his current role
at Microsoft, he spent over a decade working in the Telco and IT
industries at various capacities, from Networking Engineering to Systems
Engineering and Security Specialist.

He is a strong proponent of free and open source solutions, advocating
for the use of BSDs to connect all things.

When not hacking on computers you can catch him spending time playing with
his 1-year-old son.

SEE ALSO
   - dcasati @github
   - diegocasati @Twitter

BUGS
   Too many to list here ;}
CROSS_CLOUD_THINGS
Cross-Cloud Connectivity

What worked, what failed and lessons learned.
Ignite your curiosity by showing a real use case of an engineering exercise.
Main()

What?
Why?
Where?
How?
Deploy **Cassandra** on top of **Kubernetes** spanning two clouds.
Wait? What? Let me check online
Why?

Proactive exercise simulating Customers looking to (1) migrate or have their workload (2) spanning in two clouds
Why?

(1) **migrate** or have their workload

- For replication (e.g.: backup)
- To comply with a scheduled downtime (e.g.: keep a service running by moving data between Data Centers)
Why?

(2) **spanning** in two clouds

- To increased *availability*
- To provide load balancing
- To connect to an existing Data Center (e.g.: on premise)
Where?

AWS
Azure
How? site-to-site **VPN** between the clouds using open source components
END-TO-END TOPOLOGY - NETWORK

AWS
VPC 192.168.0.0/16
k8sDataTier 192.168.1.0/24
Kubernetes cluster
AZ us-east-1d

IPsec IKEv2 Tunnel

Microsoft Azure
Kubernetes subnet
10.0.1.0/24
NSG
Kubernetes Cluster
Dude...wait....

What???
## Common Vocabulary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jumpbox</td>
<td>A jump server is a hardened and monitored device that spans two dissimilar security zones and provides a controlled means of access between them.</td>
</tr>
<tr>
<td>VPN</td>
<td>A <strong>virtual private network (VPN)</strong> extends a private network across a public network, and enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network.</td>
</tr>
</tbody>
</table>
AW Vocab

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPC</td>
<td>A virtual private cloud (VPC) is a virtual network dedicated to your AWS account.</td>
</tr>
<tr>
<td>EIP</td>
<td>An Elastic IP address is a static, public IPv4 address designed for dynamic cloud computing.</td>
</tr>
<tr>
<td>Security Group</td>
<td>A security group acts as a virtual firewall for your instance to control inbound and outbound traffic.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.0.0/16</td>
<td>local</td>
</tr>
<tr>
<td>10.0.0/16</td>
<td>hvn1 / OpenBSD</td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td>Internet Gateway</td>
</tr>
</tbody>
</table>
### Azure Vocabulary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNET</td>
<td>Azure Virtual Network enables Azure resources to communicate with each other and the internet.</td>
</tr>
<tr>
<td>PIP</td>
<td>A PIP is a <em>public instance-level IP address</em> associated with the VM in addition to the VIP.</td>
</tr>
<tr>
<td>NSG</td>
<td>A <em>network security group (NSG)</em> contains a list of security rules that allow or deny network traffic to resources connected to Azure Virtual Networks (VNet).</td>
</tr>
<tr>
<td>UDR</td>
<td>User defined routes</td>
</tr>
</tbody>
</table>

#### Azure Vocabulary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNET</td>
<td>Azure Virtual Network enables Azure resources to communicate with each other and the internet.</td>
</tr>
<tr>
<td>PIP</td>
<td>A PIP is a <em>public instance-level IP address</em> associated with the VM in addition to the VIP.</td>
</tr>
<tr>
<td>NSG</td>
<td>A <em>network security group (NSG)</em> contains a list of security rules that allow or deny network traffic to resources connected to Azure Virtual Networks (VNet).</td>
</tr>
<tr>
<td>UDR</td>
<td>User defined routes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>Target</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.0.0/16</td>
<td>10.0.1.4</td>
<td>OpenBSD's internal NIC</td>
</tr>
</tbody>
</table>
Let’s breakdown the solution
The solution – in pieces

<table>
<thead>
<tr>
<th>Solution</th>
<th>AWS</th>
<th>Azure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>Kubernetes YAML files</td>
<td>Helm Chart</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>KOPS</td>
<td>ACS-Engine</td>
</tr>
<tr>
<td>Network</td>
<td>VPN: OpenVPN? StrongSwan? Other IPSec solution?</td>
<td></td>
</tr>
</tbody>
</table>
Cassandra
Apache Cassandra is a free and open-source distributed NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.
Kubernetes
Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.
A look at how Kubernetes fits into your infrastructure

https://www.redhat.com/en/topics/containers/what-is-kubernetes
Microsoft Azure Container Service Engine - Builds Docker Enabled Clusters

Overview

The Azure Container Service Engine (acs-engine) generates ARM (Azure Resource Manager) templates for Docker enabled clusters on Microsoft Azure with your choice of DC/OS, Kubernetes, Swarm Mode, or Swarm orchestrators. The input to the tool is a cluster definition. The cluster definition is very similar to (in many cases the same as) the ARM template syntax used to deploy a Microsoft Azure Container Service cluster.

The cluster definition file enables the following customizations to your Docker enabled cluster:

- choice of DC/OS, Kubernetes, Swarm Mode, or Swarm orchestrators
- multiple agent pools where each agent pool can specify:
  - standard or premium VM Sizes,
  - node count,
- Virtual Machine ScaleSets or Availability Sets,
- Storage Account Disks or Managed Disks (under private preview)
- Docker cluster sizes of 1200
- Custom VNET

ACS-Engine
https://github.com/Azure/acs-engine
KOPS

https://github.com/kubernetes/kops
Network
OSI Model – It’s 1999 all over again
Application
Presentation
Session
Transport
Network
Data
Physical

OpenVPN
IPSec
IPsec and its RFCs – This is HARD!
Fear not OpenIKED is here!

OpenIKED is a **FREE** implementation of the Internet Key Exchange (IKEv2) protocol which performs mutual authentication and which establishes and maintains IPsec VPN security policies and associations (SAs) between peers.

https://www.openiked.org
## The solution – in pieces

<table>
<thead>
<tr>
<th>Solution</th>
<th>AWS</th>
<th>Azure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>Kubernetes YAML files</td>
<td>Helm Chart</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>KOPS</td>
<td>ACS-Engine</td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td><strong>VPN: OpenIKED</strong></td>
</tr>
</tbody>
</table>
What failed?
#FAIL!

<table>
<thead>
<tr>
<th>Attempt</th>
<th>Approach</th>
<th>Why it failed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Site-to-site VPN between AWS’s Virtual Gateway and Azure’s VPN Gateway</td>
<td>Tunnels did not sync and Hard to troubleshoot</td>
</tr>
<tr>
<td>#2</td>
<td>GRE tunnel between FreeBSD VMs</td>
<td>Site-to-site VPN between AWS’s Virtual Gateway and Azure’s VPN Gateway</td>
</tr>
<tr>
<td>#3</td>
<td>GIF tunnel between FreeBSD VMs</td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>IPsec tunnel on FreeBSD (need either GRE or GIF...)</td>
<td></td>
</tr>
</tbody>
</table>
What worked?
## IT_WORKS!

<table>
<thead>
<tr>
<th>Attempt</th>
<th>Approach</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>SSH-based VPN</td>
<td>For testing only. Overhead is too high. Refer to ssh(1) for details</td>
</tr>
<tr>
<td>#2</td>
<td>VM to Gateway service (both AWS and Azure)</td>
<td>Works for NVAs such as Cisco ASA’s</td>
</tr>
<tr>
<td>#3</td>
<td>IKEv2 tunnel with OpenBSD</td>
<td>IPsec. Easiest OSS option</td>
</tr>
</tbody>
</table>
$ ssh -o "VerifyHostKeyDNS ask" host.example.com

[...]
Matching host key fingerprint found in DNS.
Are you sure you want to continue connecting (yes/no)?

See the VerifyHostKeyDNS option in ssh_config(5) for more information.

---

**SSH-BASED VIRTUAL PRIVATE NETWORKS**

ssh contains support for Virtual Private Network (VPN) tunnelling using the tun(4) network pseudo-device, allowing two networks to be joined securely. The sshd_config(5) configuration option **PermitTunnel** controls whether the server supports this, and at what level (layer 2 or 3 traffic).

The following example would connect client network 10.0.50.0/24 with remote network 10.0.99.0/24 using a point-to-point connection from 10.1.1.1 to 10.1.1.2, provided that the SSH server running on the gateway to the remote network, at 192.168.1.15, allows it.

On the client:

```
# ssh -f -w 0:1 192.168.1.15 true
# ifconfig tun0 10.1.1.1 10.1.1.2 netmask 255.255.255.252
# route add 10.0.99.0/24 10.1.1.2
```

On the server:

```
# ifconfig tun1 10.1.1.2 10.1.1.1 netmask 255.255.255.252
# route add 10.0.50.0/24 10.1.1.1
```

Client access may be more finely tuned via the `/root/.ssh/authorized_keys` file (see below) and the **PermitRootLogin** server option. The following entry would permit connections on tun(4) device 1 from user `jane` and on tun device 2 from user `joh": if PermitRootLogin is set to `forced-commands-only`:

```
tunnel="1",command="sh /etc/netstart tun1" ssh-rsa ... jane
tunnel="2",command="sh /etc/netstart tun2" ssh-rsa ... joh
```

Since an SSH-based setup entails a fair amount of overhead, it may be more suited to temporary setups, such as for wireless VPNs. More permanent VPNs are better provided by tools such as ipsecctl(8) and isakmpd(8).
The solution – in pieces

<table>
<thead>
<tr>
<th>Solution</th>
<th>AWS</th>
<th>Azure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>Kubernetes YAML files</td>
<td>Helm Chart</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>KOPS</td>
<td>ACS-Engine</td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td>VPN: OpenIKED on OpenBSD</td>
</tr>
</tbody>
</table>
Why OpenBSD?

1. A **FREE**, multi-platform 4.4BSD-based UNIX-like operating system.
2. **Proactive security** (strlcpy, strlcat, W^X, privilege separation, ...)
3. Many of our day-today tools in the base install (by default):
   - tmux, nvi, OpenSSH, mg (emacs-like), tcpdump, ...
4. Infrastructure tools:
   - OpenIKED – IKEv2 Daemon
   - OpenBGPD – BGP daemon
   - OpenNTPD – NTP daemon
   - OpenOSPFD – OSPF daemon
   - OpenSMTPD – SMTP daemon
   - Relayd – L3/7 Load balancer
   - httpd – HTTP daemon
   - ACME-client – ACME certificate client
   - Rebound – DNS proxy
   - PF - firewall
Why OpenBSD?

Problem:
- Vanilla image available on AWS or Azure.
- Option to use Esdenera Firewall 3 (NVA based on OpenBSD).
- Our current documentation is not working – Needs TLC.

Solution:
- Bake your own image.
- Based on scripts from core OpenBSD devs (folks from Esdenera): qemu, Makefile et al.
- While doing this, 3 PRs were opened to fix issues (AWS and documentation).
- All files on Github:
  - [https://github.com/dcasati/cloud-openbsd](https://github.com/dcasati/cloud-openbsd)
  - [https://github.com/dcasati/ports-azure](https://github.com/dcasati/ports-azure)
END-TO-END TOPOLOGY - NETWORK
AWS

1. Create a VPC with a large network (e.g.: 192.168.0.0/16)
2. Carve 3 subnets (k8sDataTier, Management, VPN)
3. Create the OpenBSD VM with two NICs
   a) For each NIC disable source/dest check
4. Add the route to Azure (e.g.: 10.0.0.0/8)
5. Allow traffic on the Security Groups (ports 500, 4500 UDP)
6. Attach an Elastic IP to the OpenBSD interface on the VPN subnet.
7. Configure OpenIKED.

<table>
<thead>
<tr>
<th>Destination</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.0.0/16</td>
<td>local</td>
</tr>
<tr>
<td>10.0.0.16</td>
<td>hvn1 / OpenBSD</td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td>Internet Gateway</td>
</tr>
</tbody>
</table>
1. Create a VNet with a large network (e.g.: 10.0.0.0/8)
2. Carve 3 subnets (k8sDataTier, Management, VPN)
3. Create the OpenBSD VM with two NICs (via Azure CLI)
   a) For each NIC enable IP forwarding
4. Add the route to Azure (e.g.: 192.168.0.0/16) on the UDR
5. Allow traffic on the Security Groups (ports 500, 4500 UDP)
6. Attach a Public IP to the OpenBSD interface on the VPN subnet.
7. Configure OpenIKED.
OpenBSD

Configuration
/etc/iked.conf

local_gw = "51.143.95.27"
remote_gw = "34.233.91.14"
local_net = "10.0.0.0/16"
remote_net = "192.168.0.0/16"
kops_net = "100.64.0.0/10"
state = "active"

ikev2 $state ipcomp esp \n    from $local_gw to $remote_gw \n    from $local_net to $remote_net peer $remote_gw \n    psk "1BigSecret"

ikev2 $state ipcomp esp \n    from $local_gw to $remote_gw \n    from $kops_net to $remote_net peer $remote_gw \n    psk "1BigSecret"
OpenBSD

Commands

# rcctl enable iked
# rcctl start iked
# sysctl –w net.inet.ip.forwarding=1

To check if Ipsec is working:
# ipsecctl –sa
Lessons Learned
Pressing all of the right buttons

Default VPC route table missing entries
Security Group
VPC ACLs
source/dest check
IKEDv2 configuration
net.inet.ip.forwarding=0
OS-level firewall
IP Forwarding
Route Table
NSG:Inbound
Resource Group
OS
EC2
VPC

Problem Statement
No traffic
PLEASE
REFRESH
MY MIND
Our goal: Deploy **Cassandra** on top of **Kubernetes** spanning two clouds
END-TO-END TOPOLOGY - KUBERNETES

AWS

VPC 192.168.0.0/16

VPC

k8sDataTier 192.168.1.0/24

Kubernetes cluster

AZ us-east-1d

IPsec IKEv2 Tunnel

Microsoft Azure

Kubernetes subnet 10.0.1.0/24

NSG

Kubernetes Cluster

Kubernetes cluster
END-TO-END TOPOLOGY WITH CASSANDRA!
END-TO-END TOPOLOGY - NETWORK

<table>
<thead>
<tr>
<th>Destination</th>
<th>Target</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.0.0/16</td>
<td>local</td>
<td></td>
</tr>
<tr>
<td>10.0.0/16</td>
<td>hvn1 / OpenBSD</td>
<td></td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td>Internet Gateway</td>
<td></td>
</tr>
<tr>
<td>192.168.0.0/16</td>
<td>10.0.1.4</td>
<td>OpenBSD's internal NIC</td>
</tr>
<tr>
<td>100.64.0.0/10</td>
<td>10.0.1.4</td>
<td>KOPS Network</td>
</tr>
</tbody>
</table>
Demo
Next Steps

- Hashicorp’s Terraform Templates
- Hashicorp’s Packer to build the images
- Highly Available Solutions
Key takeaways

A better view of the design choices when connecting public clouds

Understand that failures will happen – and that’s ok

Reach out for help. Talk to the community
Q&A
Let’s talk

diegocasati @twitter

https://github.com/dcasati/cross-cloud-vpn