VPC Lattice: yet another connectivity option or a game-changer?

12 April 2024 - 4 min. read

Advanced Networking

VPC Lattice

"Live as if you were to die tomorrow. Learn as if you were to live forever"

- *Mahatma Gandhi*

I like learning new things, and I am lucky because a significant part of my work involves learning new technologies and evaluating how to use them to solve new and existing problems.

In the past few years, we described many connectivity options to connect our workloads running on AWS (Transit Gateways, Load balancing, Shared VPCs, PrivateLink and Endpoint Services, and VPC peering...)

Last year, **VPC Lattice** reached General Availability, offering a new option for us to explore. You may be wondering why we should learn how to use another service when we already have so many options to choose from... Well, let's find out!

In this article, we'll see that we needed to add another thing... Let's see what this service offers, how it differs from the other connectivity options we already know, and a simple use case.

So, without any further ado, let's dig in!

Where VPC Lattice stands

As we saw in previous articles, microservices are an architectural approach that allows developers to build and deploy software faster. How

single applications now consist of numerous indivic
distributed in different AWS accounts and must cor
various technologies can be used together: Lambda

We must face tasks like service discovery, traffic rou
detailed metrics. There are scenarios where a strict
authorization policy is complex to implement using

Think about allowing communication only between two microservices in different AWS accounts, blocking every other traffic without implementing custom solutions and authentication.

VPC Lattice aims to help organizations overcome these challenges while keeping a secure and manageable approach.

It enables developers to **manage microservices definition, authorization, and configuration, while administrators can manage infrastructure, network, communication policies, and governance;** we'll see more in the following sections.

VPC Lattice Components

VPC Lattice has some key components:

- **Service**: a VPC Lattice service represents an application and its resources: computational (such as lambdas, EKS pods, EC2 instances), listeners (port and protocols involved), and routing rules to address traffic (weighted routing, path routing, and so on). The application team typically manages these aspects and can be autonomously defined. A service can be shared using AWS Resource Access Manager with other AWS accounts, even outside an organization.
- **Service Network**: a service network is a new concept because it's a logical grouping of services that facilitates connectivity. It can also be shared using AWS Resource Access Manager. This kind of resource is usually defined by infrastructure or network administrators.
- **Service Directory:** the list of service networks that can be used, shared with AWS RAM
- **Auth Policies:** IAM documents that can define access to resources. Auth policies differ from IAM policies because they aren't attached to users, groups, or roles but are used with services and service networks. The

or other services access to the attached service.
conditions, such as

```
vpc-lattice-svcs:RequestMethod
```

to filter allowed methods. For a complete list, you c
documentation.

Let's see which steps are required to set up VPC Lattice.

Putting all together

Let's briefly look at the roles and steps involved in setting up the communication.

1. The infrastructure admin defines a **Service Network**
2. The infrastructure admin shares the Service Network using **AWS RAM**
3. The service owner defines a **VPC Lattice Service** and its authorization policies
4. The service owner associates the **Service** with the Service Network
5. The infrastructure admin **associates** the Service Network with the **VPCs**

As you can see, there is a certain degree of independence in managing services, so dependencies between teams and operations are reduced.

A practical example of VPC Lattice usage

For our example, we will use the sample infrastructure and applications from this URL and analyze the key components deployed on the AWS Console.

You can deploy the reference architecture using the instructions contained in the *ReadMe* file in the repository.

You can also use three different VPCs belonging to a single account to ease the deployment process (as we'll do in this article).

You'll end with this sample infrastructure:

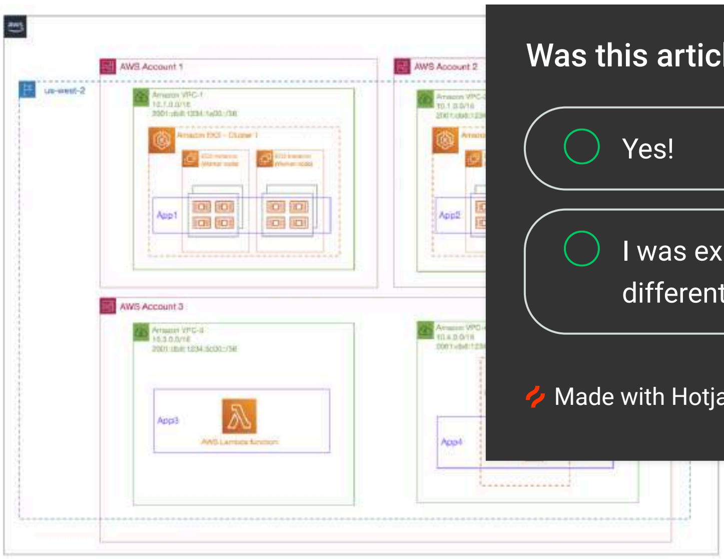


Image source: https://github.com/aws-samples/build-secure-multi-account-vpc-connnectivity-applications-with-amazon-vpc-lattice

In our example, App1 and App2 will need to communicate bidirectionally, and App3 will be consumed by App1, App2, and App4.

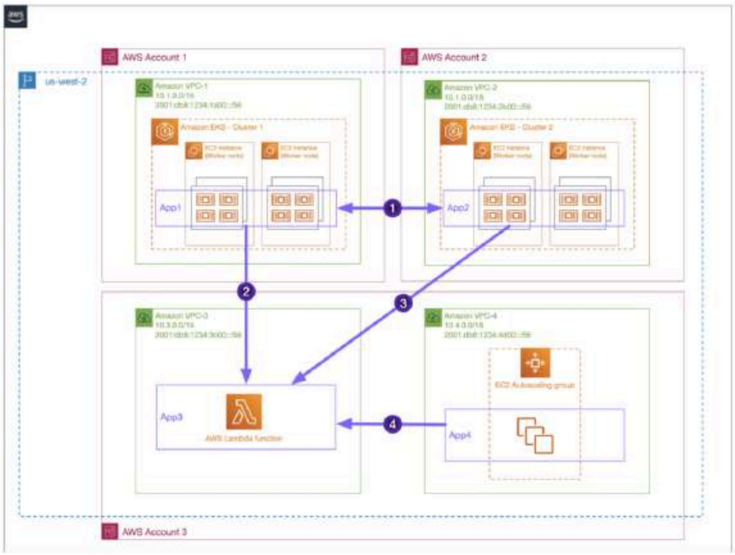


Image source: https://github.com/aws-samples/build-secure-multi-account-vpc-connnectivity-applications-with-amazon-vpc-lattice

The deployment process will take about 10 - 15 minutes, and you will see that our microservices can communicate.

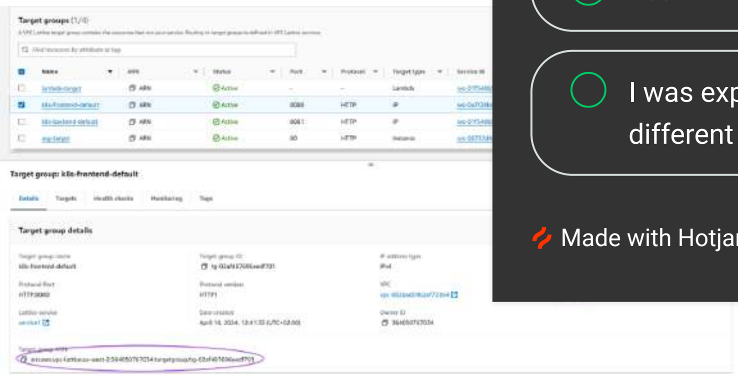
Bug Alert: there's a missing step in the deployment instructions; these two lines will make the deployment fail:

```
export TARGETCLUSTER1={TARGET_GROUP_ARN}
export TARGETCLUSTER2={TARGET_GROUP_ARN}
```

You need to change these exports using values fror

For TARGETCLUSTER1, use the k8s-frontend-defau

the arn for k8s-backend-default. This is a sample fr



Target groups (1/4)

A VPC Lattice target group contains the resources that run your service. Routing to target groups is defined in VPC Lattice services.

Find resources by attribute or tag

	Name	ARN	Status	Port	Protocol	Target type	Service ID
□	lambda-target	ARN	Active	-	-	Lambda	svc-[illegible]
☑	k8s-frontend-default	ARN	Active	8080	HTTP	IP	svc-[illegible]
□	k8s-backend-default	ARN	Active	8081	HTTP	IP	svc-[illegible]
□	asg-target	ARN	Active	80	HTTP	Instance	svc-[illegible]

Target group: k8s-frontend-default

Details | Targets | Health checks | Monitoring | Tags

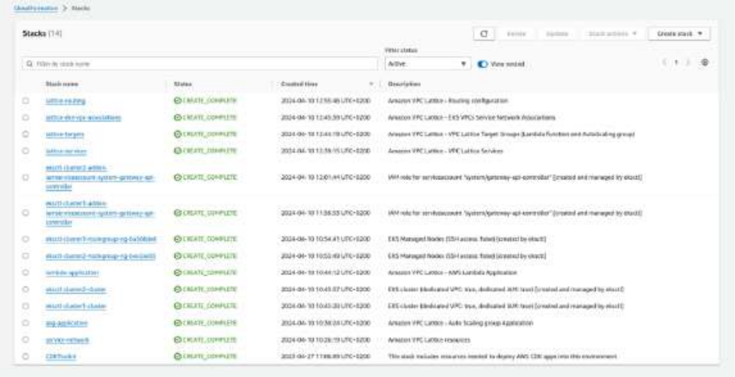
Target group details

Target group name k8s-frontend-default	Target group ID tg-[illegible]	IP address type IPv4
Protocol: Port HTTP:8080	Protocol version HTTP1	VPC vpc-[illegible]
Lattice service service1	Date created April 10, 2024, 12:41:33 (UTC+02:00)	Owner ID 364050767034

Target group ARN

arn:aws:vpc-lattice:eu-west-2:364050767034:targetgroup/tg-[illegible]

When the CloudFormation deployment finishes, you should have these CloudFormation stacks:



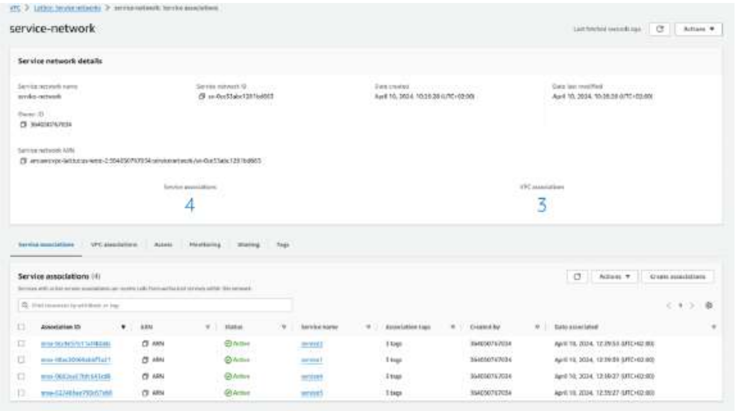
CloudFormation > Stacks

Stacks (14)

Filter by stack name

	Stack name	Status	Created time	Description
○	lattice-routing	CREATE_COMPLETE	2024-04-10 12:55:48 UTC+0200	Amazon VPC Lattice - Routing configuration
○	lattice-eks-vpc-associations	CREATE_COMPLETE	2024-04-10 12:45:39 UTC+0200	Amazon VPC Lattice - EKS VPCs Service Network Associations
○	lattice-targets	CREATE_COMPLETE	2024-04-10 12:44:19 UTC+0200	Amazon VPC Lattice - VPC Lattice Target Groups (Lambda Function and AutoScaling group)
○	lattice-services	CREATE_COMPLETE	2024-04-10 12:38:15 UTC+0200	Amazon VPC Lattice - VPC Lattice Services
○	eksctl-cluster2-addon-iamserviceaccount-system-gateway-api-controller	CREATE_COMPLETE	2024-04-10 12:01:44 UTC+0200	IAM role for serviceaccount "system/gateway-api-controller" [created and managed by eksctl]
○	eksctl-cluster1-addon-iamserviceaccount-system-gateway-api-controller	CREATE_COMPLETE	2024-04-10 11:58:53 UTC+0200	IAM role for serviceaccount "system/gateway-api-controller" [created and managed by eksctl]
○	eksctl-cluster1-nodegroup-ng-[illegible]	CREATE_COMPLETE	2024-04-10 10:54:41 UTC+0200	EKS Managed Nodes (SSH access: false) [created by eksctl]
○	eksctl-cluster2-nodegroup-ng-[illegible]	CREATE_COMPLETE	2024-04-10 10:53:49 UTC+0200	EKS Managed Nodes (SSH access: false) [created by eksctl]
○	lambda-application	CREATE_COMPLETE	2024-04-10 10:44:12 UTC+0200	Amazon VPC Lattice - AWS Lambda Application
○	eksctl-cluster2-cluster	CREATE_COMPLETE	2024-04-10 10:43:37 UTC+0200	EKS cluster (dedicated VPC: true, dedicated IAM: true) [created and managed by eksctl]
○	eksctl-cluster1-cluster	CREATE_COMPLETE	2024-04-10 10:43:20 UTC+0200	EKS cluster (dedicated VPC: true, dedicated IAM: true) [created and managed by eksctl]
○	asg-application	CREATE_COMPLETE	2024-04-10 10:38:24 UTC+0200	Amazon VPC Lattice - Auto Scaling group Application
○	service-network	CREATE_COMPLETE	2024-04-10 10:28:19 UTC+0200	Amazon VPC Lattice resources
○	CDKToolkit	CREATE_COMPLETE	2023-06-27 11:06:09 UTC+0200	This stack includes resources needed to deploy AWS CDK apps into this environment

Let's see the resources created to describe them. First, the service network:



VPC > Lattice: Service networks > service-network: Service associations

service-network

Service network details

Service network name service-network	Service network ID sn-0cc53abc1281bd663	Date created April 10, 2024, 10:28:28 (UTC+02:00)	Date last modified April 10, 2024, 10:28:28 (UTC+02:00)
Owner ID 364050767034			

Service network ARN

arn:aws:vpc-lattice:eu-west-2:364050767034:servicenetwork/sn-0cc53abc1281bd663

Service associations

4

VPC associations

3

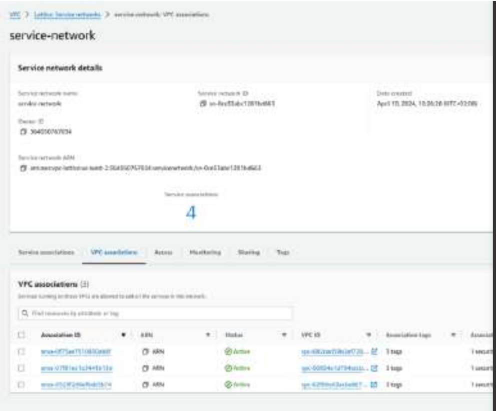
Service associations | VPC associations | Access | Monitoring | Sharing | Tags

Service associations (4)

Find resources by attribute or tag

□	Association ID	ARN	Status	Service name	Association tags	Created by	Date associated
□	snsa-[illegible]	ARN	Active	service2	1 tags	364050767034	April 10, 2024, 12:39:53 (UTC+02:00)
□	snsa-[illegible]	ARN	Active	service1	1 tags	364050767034	April 10, 2024, 12:39:59 (UTC+02:00)
□	snsa-[illegible]	ARN	Active	service4	1 tags	364050767034	April 10, 2024, 12:38:27 (UTC+02:00)
□	snsa-[illegible]	ARN	Active	service3	1 tags	364050767034	April 10, 2024, 12:38:27 (UTC+02:00)

As you can see, four services are associated; you can also see VPCs associated with the service network:



service-network

Service network details

April 10, 2024, [illegible]

4

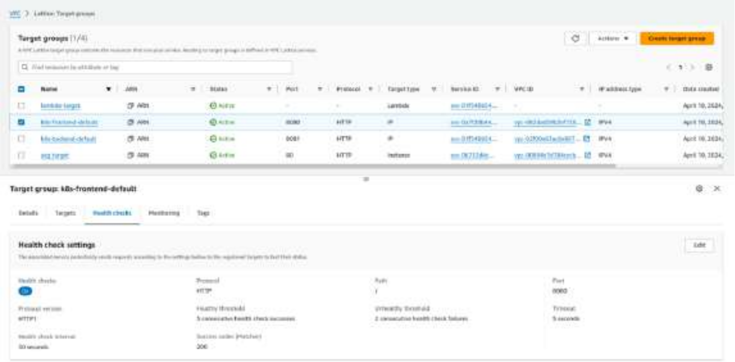
Service associations | VPC associations | Access | Monitoring | Sharing | Tags

VPC associations (3)

Association ID	ARN	Status	VPC ID	Association tags
[illegible]	ARN	Active	[illegible]	1 tags
[illegible]	ARN	Active	[illegible]	1 tags
[illegible]	ARN	Active	[illegible]	1 tags

Now VPC Lattice target groups:

Our Lambda, Autoscaling group, and EKS are added as targets, and like ELB target groups, you can see metrics and define health checks.



Target groups (1/4)

Name	ARN	Status	Port	Protocol	Target type	Service ID	VPC ID	IP address type	Date created
lambda-target	ARN	Active	-	-	Lambda	[illegible]	-	-	April 10, 2024,
k8s-frontend-default	ARN	Active	8080	HTTP	IP	[illegible]	[illegible]	IPV4	April 10, 2024,
k8s-backend-default	ARN	Active	8081	HTTP	IP	[illegible]	[illegible]	IPV4	April 10, 2024,
asg-target	ARN	Active	80	HTTP	Instance	[illegible]	[illegible]	IPV4	April 10, 2024,

Target group: k8s-frontend-default

Details | Targets | Health checks | Monitoring | Tags

Health check settings

Health checks: On

Protocol: HTTP

Path: /

Port: 8080

Protocol version: HTTP1

Healthy threshold: 5 consecutive health check successes

Unhealthy threshold: 2 consecutive health check failures

Timeout: 5 seconds

Health check interval: 30 seconds

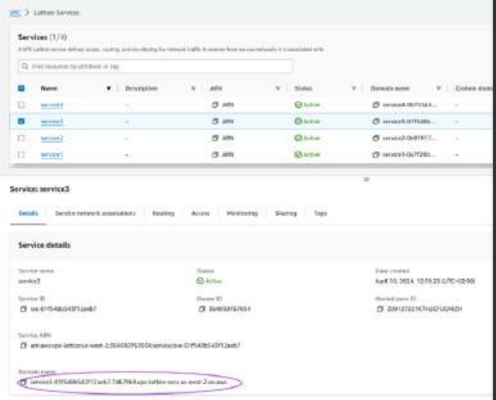
Success codes (Matcher): 200

Note that to register our targets managed by EKS clusters, we had to deploy the Gateway API Controller Class into the cluster and register the service using the configuration deployed with the commands:

```
kubectl config use-context cluster1
kubectl apply -f ./vpc-lattice/routes/frontend-export.yaml

kubectl config use-context cluster2
kubectl apply -f ./vpc-lattice/routes/backend-export.yaml
```

Last but not least, let's see the defined services. You can access them using the DNS domain name defined by VPC Lattice



Lattice Services

Services (1/4)

Name	Description	ARN	Status	Domain name
service4	-	ARN	Active	[illegible]
service3	-	ARN	Active	[illegible]
service2	-	ARN	Active	[illegible]
service1	-	ARN	Active	[illegible]

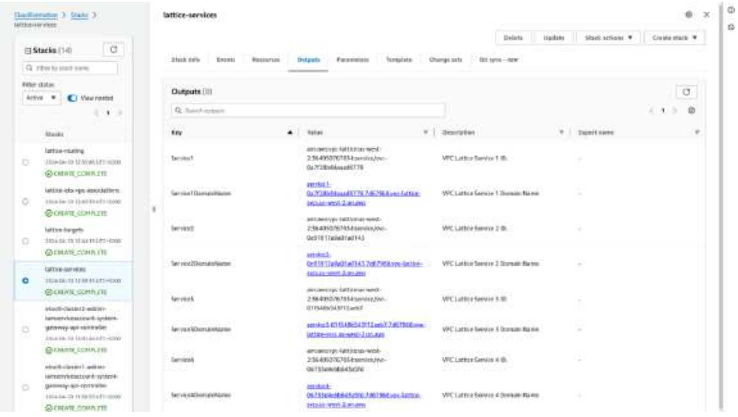
Service: service3

Details · Service network associations · Routing · Access · Monitoring · Sharing · Tags

Service details

Service name: service3
Status: Active
Date created: April 10, 2024, 12:59:25 (UTC+02:00)
Service ID: [illegible]
Owner ID: [illegible]
Service ARN: [illegible]
Domain name: [illegible]

You can find the complete list of domain names defined by looking at the Outputs of the CloudFormation stack named "lattice-services":



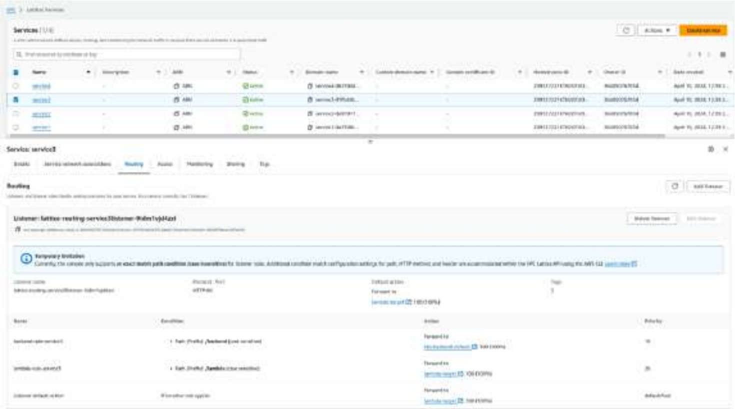
lattice-services

Stack info · Events · Resources · Outputs · Parameters · Template · Change sets · Git sync - new

Outputs

Key	Value	Description	Export name
Service1	[illegible]	VPC Lattice Service 1 ID.	-
Service1DomainName	[illegible]	VPC Lattice Service 1 Domain Name	-
Service2	[illegible]	VPC Lattice Service 2 ID.	-
Service2DomainName	[illegible]	VPC Lattice Service 2 Domain Name	-
Service3	[illegible]	VPC Lattice Service 3 ID.	-
Service3DomainName	[illegible]	VPC Lattice Service 3 Domain Name	-
Service4	[illegible]	VPC Lattice Service 4 ID.	-
Service4DomainName	[illegible]	VPC Lattice Service 4 Domain Name	-

Each service also has its routing; in this case, you can see that /backend path for service3 forwards to cluster2, while /lambda forwards requests to the lambda application:



Services (1/4)

Service: service3

Routing

Listener: lattice-routing-service3listener-[illegible]

Temporary limitation

Name	Condition	Action	Priority
backend-rule-service3	Path (Prefix) /backend	Forward to [illegible]	[illegible]
lambda-rule-service3	Path (Prefix) /lambda	Forward to [illegible]	[illegible]
Listener default action	If no other rule applies	Forward to [illegible]	default

Now that you have a deployed infrastructure, feel free to experiment and modify the behaviour of your application and environment (why don't you try IAM authentication between services?).

This is only a starting point, and you can also find an excellent AWS workshop with a good walkthrough here.

Last but not least: is VPC Lattice the right t

The correct answer is, as always: "It depends". Let's
impact on our design.

The first is the mandatory **one-to-one association b**
Network. Once you define an association, you cann
other service networks, so you need to plan the imp

You cannot share services between different region
regional construct: you'll need to use a hybrid approach, resorting to traditional" connectivity methods, like Transit Gateway Peering and VPC Peering.

On the other hand, there are great **benefits**: you can free service owners to configure and offer by themselves their components while keeping centralized control in a complex environment.

We didn't have a look at IAM policies, but **implementing IAM Authentication and Authorization is the key to a secure microservice implementation**.

VPC Lattice also offers a complete service mesh solution because it can link EC2, EKS, ECS, and Lambda workloads.

Cost is another important factor: **as your service mesh grows, costs increase proportionally**.

You pay 0.025$/hour for each service. If you have 10 services you pay 182.5$/month. A different approach can have a smaller impact on AWS billing for complex architectures with hundreds of microservices.

That's all for today!

Microservices aren't simple: they can increase the overall complexity of the architecture and require good planning and management over time.

Have you already had the chance to experiment with VPC Lattice? What are your thoughts about this new approach?

Let us know in the comments!

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN t
in designing, implementing, and managing complex
advanced services on AWS. Before being writers, w
with AWS services since 2007. We are hungry reade
seekers. On Proud2beCloud, we regularly share our
insights, in-depth news, tips&tricks, how-tos, and m
the discussion!

Was this article useful for you?

○ Yes!

○ I was expecting something different

Made with Hotjar

Damiano Giorgi

Ex on-prem systems engineer, lazy and prone to automating boring tasks. In constant search of technological innovations and new exciting things to experience. And that's why I love Cloud Computing! At this moment, the only "hardware" I regularly dedicate myself to is that my bass; if you can't find me in the office or in the band room try at the pub or at some airport, then!