ASSESSMENT.

Sjoerd Santema | 2018

SENTIA

TABLE OF CONTENT

FOREWORD

First of all, I would like to thank you for giving me the opportunity to show my motivation and eagerness to become an AWS engineer.

In this document I'll showcase you how I would design an AWS environment based on the provided requirements. In doing so I will try to not only repeat what I've learned during the (very awesome) Sentia AWS Academy but extend my knowledge by doing research on different approaches and log these attempts to give you a clear insight in my *modus operandi*.

SCENARIO

You have a client which is a web shop. They have a standard Magento setup with MySQL on a single VPS at their current provider and want to bring this to AWS.
The external software (Magento) is managed, developed and maintained by an external party.

1. Design an architecture which incorporates the following client needs. The clients know that his whole setup is going to evolve constantly. Include a template for programmable infrastructure for your design.
- Needs to be scalable and flexible
- Needs to have low latency for SEO purposes
- Needs to be cost effective

2. After releasing the new architecture, business takes on, and the client decides to add customer reviews.
- Do you need to alter your architecture? And if so, how?

3. At some point, one of the customer employees is getting very good at creating vlogs, and the client wants to give customers the opportunity to upload videos with their reviews. They want to store the thumbnails and videos for later processing, and they want to show thumbnails of the videos underneath the product pages.
- Alter your architecture to process and store these videos.

4. At some point, some clients uploaded non-compliant video's, and which created a huge marketing issue. The client now wants to screen the uploaded video's before putting them online, but with minimal costs.
- Alter your architecture to be able to screen and process these video's.

DELIVERABLES

1. a time log of your activities and the road to the solutions
2. per question (1 to 4) a complete answer, including designs and reasoning of chosen components and configurations
3. if applicable including template files

APPROACH

Before I start writing even a single line of code I want to create a clear overview of what is needed to complete this assignment and how the later questions could impact the design I make in the first question. There are two parts in this assessment: 1) the best architecture for this Magento application and 2) additional projects to allow for video uploads by customers and later reviewing the posted videos.

To prevent myself from chasing a red herring and spending a lot of hours (paid by the customer) on a design invalidated by the second part of the assignment, I need to first decide on how I roughly want to design the video reviewing process.

My first thoughts are to create two different buckets, one for video's that are up for review and a second for reviewed video's. Then creating some sort of reviewing process assisted by Lambda functions. But before being the hammer and that only sees AWS nails, I want to research if there's an existing solution for this in Magento. Why spend precious hours on re-inventing this video review wheel?

Googling "Magento video review" quickly brought me to the following websites:
https://www.fmeextensions.com/video-testimonials.html
https://ecommerce.aheadworks.com/magento-extensions/video-testimonials.html

The second website gave me insights in some very important reasons as why you would use this kind of extension instead of a tailor-made AWS solution. First of all, it nicely integrates into Magento, something the AWS solution wouldn't easily be able to do. The AWS solution would require altering the application which needs a Magento developer to assist. Not to mention the maintenance. There's another factor the client needs to be aware of and that is the effect Youtube customer reviews van have on the SEO of the site.

Implementing an AWS driven solution could force us straight into the pickle. Not only do we need to find a way for authorised users to upload movies to an S3 bucket, we would also need to store the link to the movie in the database and link it to the appropriate product. All in all a lot of work which has a lot of potential of spending hours and hours without bringing the best solution to the customer.

To make sure I'm presuming something that actually makes sense, I'll call a friend who's working as an Magento developer. Better be safe then spending hours on a lost cause.

My suggestion would be to talk to the customer to find out what his/her motives are and what solution would fit best. So, let's carry on and proceed with the first part of the assignment!

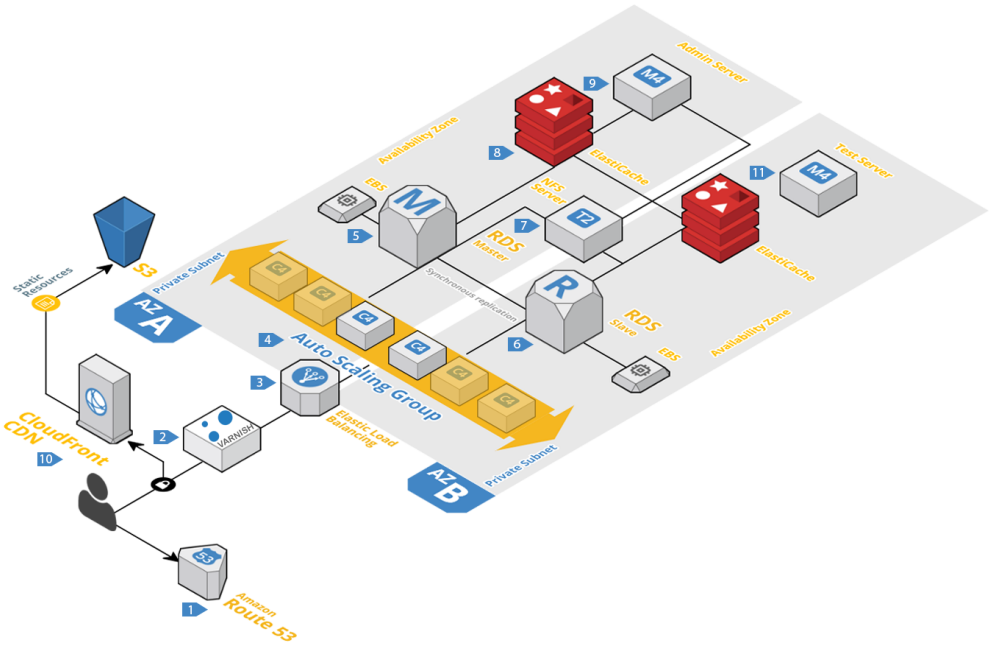
QUESTION 1

After giving it some thought I've decided to start off with the qualities required by the customer. These are clearly defined in the assignment. The environment needs to have the following qualities:

1. Scalable and flexible
2. Low latency (for SEO purposes)
3. Cost effective
4. Designed in templates

The application is Magento 2, one of the world's most popular opensource e-commerce platforms. On the internet one can find an abundance of extensions, documentation and best practices. It seems logical to start looking for information on the internet.

In my previous job at Byte (a hosting company focused om Magento) I stumbled upon the offerings of mgt-commerce.com. Their proposition is recognized in the industry as one of the best on AWS. Browsing on their website I've found this design:



https://www.mgt-commerce.com/cms/auto-scaling-plans-and-pricing

This page shows you how MGT has designed their Magento AWS platform. There are some peculiarities in this design. One of them is the NFS server that seems to stand in between both AZ's and then there are the admin server and test server that are hidden behind Elasticache (instead of right next to the other application servers). One private subnet is shown but all other subnets seem to be missing. The admin server isn't in an auto-scaling group while this could bring some real advantages. Varnish seems to be served from one instance but it's unclear in what AZ it's located and there seems to be a load balancer missing. From other projects I've learned Varnish has so be served over two instances in two AZ's (for HA) combined with two load balancers. Definitely something worth investigating and it's good to take this diagram with a grain of salt.

REQUIREMENTS

The aforementioned wished-for qualities should be reflected in the requirements. We do have to make some assumptions, in particular about the RTO, RPO and DR. Some 'applied brain' should tell us that e-commerce websites can have some serious amounts of customer 'stampedes', especially if the company invests in tv-commercials. This needs to be taken in account when designing the architecture.

In a real-life situation I would consult the customer on all requirements and advice on the impact it has on costs. I would most definitely talk to the third-party developer to better understand their current way of working and how our design would fit in their needs. If we can save them many hours of manual deployments, this indirectly would be in the advantage of the customer and could fuel an intensification of the partnership with this third party.

Let's assume, since we're talking about a fictive customer, the customer wants the website to be highly available but isn't willing to spend the money on a fault-tolerant environment. Disaster Recovery is a nice to have, but not something we should focus on. This customer wants his/her website to be fast and available at all times (just not when disaster strikes). Lacking any of both could result in some serious SEO penalties and subsequent lost revenue. DDoS mitigation is a serious concern as competition is fierce and our customer is afraid to fall victim to other companies with nefarious intents.

DESIGN

After reviewing the current specifications and architecture I've concluded to start off with a basic design in two availability zones in the eu-west-1 region. The AWS CLI tells me there are three AZ's in this region so depending on the number of application servers we're going to deploy, we have the possibility to expand to three AZ's.

```
aws ec2 describe-availability-zones --region eu-west-1
```

A contributing factor is to choose this region is that it offers all major AWS services we need for this assignment. I've read this article prior to this assignment and bookmarked it:
https://www.concurrencylabs.com/blog/choose-your-aws-region-wisely/
Ireland looks to be the leader of the pack regarding the introduction of new services. Let's stick with Ireland first.

Cost is important for this customer and it can differ per region. Let's make a comparison after we've decided what components we will use.

SKELETON

We've decided to use the two AZ's in eu-west-1. A CDN for DDoS mitigation and caching of static content can be added later, so let's focus on what's inside the VPC first.

To start off with we need a load balancer. AWS offers three kinds of load balancers, classic, ELB and ELBv2. I've chosen for the ELB since it offers some nice features like the integration with WAF (which could be useful when the customer grows), the ability for path-based routing (which we e.g. could use for the admin server), the integration with ECS (which we might use now or in the future) and the ability to use the unique trace identifier in debugging performance issues. Complementary it's HTTP/2 native which can positively impact the performance of Magento. AWS takes care of the availability of this service. So ELBv2 it is! I've learned from the academy (read: the hard way) you can't just upgrade the load balancer, working in Cloudformation means it needs replacing the load balancer altogether.

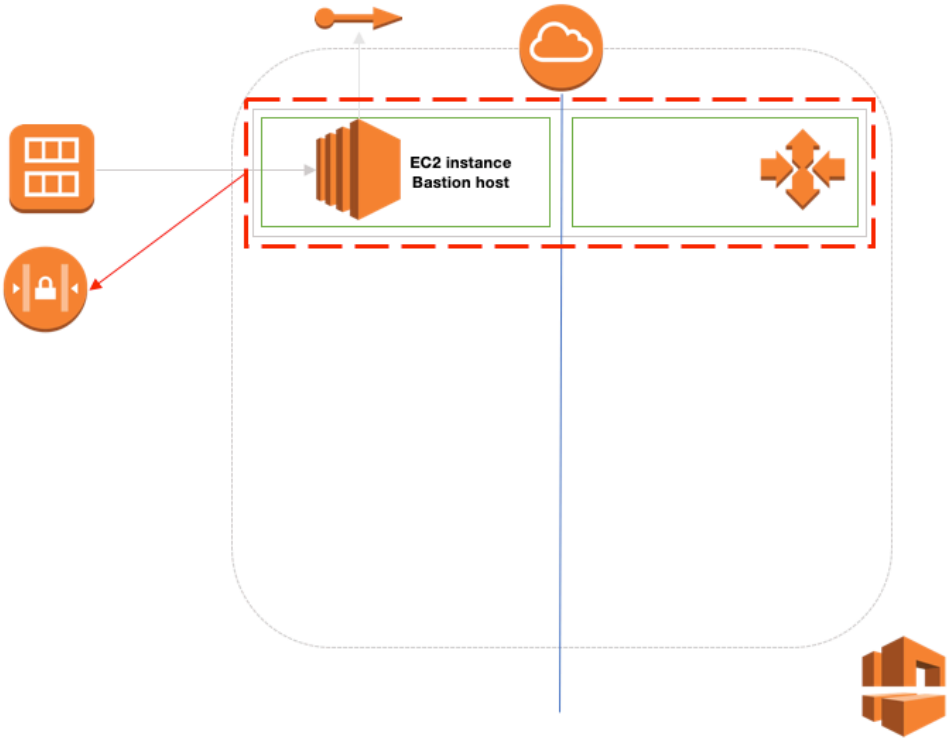
Being able to debug every step you're taking is crucial when building any environment. Without it you'll spend lots of customers time on eliminating one of the many factors. With this in mind I've decided to not only make a design but a step-by-step guide on how I want to build this setup.

So, before we even start with the load balancer and the rest, let's start with a bastion host so we can SSH into any EC2 instance and see what's going on. The VPC has been created and so has the attached-IGW.

We need to select an AMI for the bastion host. Its functionality will be very limited (it's just a bastion host after all), so let's use an AMI of the shelf. Obviously, we first need to create a keypair to be able to login to it. Now let's create a security group and subnet to launch the bastion host in. But let's think about what this bastion host does. It's our single point of SSH entry so we don't want to be locked out if one AZ becomes unavailable. A cost-efficient way to do this is by creating an auto-scaling group for just one instance and let it span two or more subnets in two or more AZ's. And while we're at it lets attach an EIP as well so we know where to find it.

Of course, we need to add a security group and nACL to secure it from the base up. Let's write the template, add an auto-scaling group (and launch config) and see if we can SSH into our newly created bastion host.

The design now looks something like this:



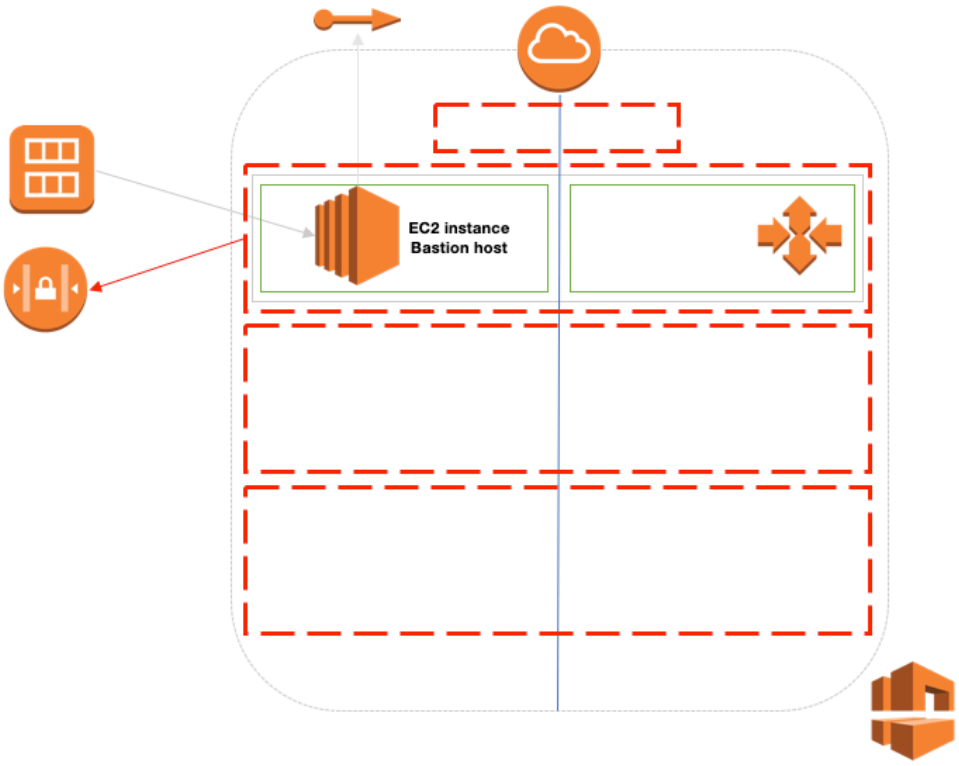
For this assessment I've skipped rewriting this part of the template considering I've spent countless hours creating the exact same skeleton template for the Sentia Academy course. I do want to show the reader how I would evaluate every component to match it with the set-out requirements. We now have a highly-available secure entry point.

ONE BASTION HOST LATER

After deploying the first version of the skeleton template I can login to the bastion host. Great, now we can focus on the rest of the skeleton template.

We now need to spawn more subnets to accommodate the other components. Let's start with creating the security groups. The bastion host security group will need to access to the application servers security group. We create the application servers security group and thereafter the database security group. The database security group allows ingress from the application servers security group and the application servers security group allows ingress from the bastion host security group. But let's not forget about the load balancer security group! This group needs to take ingress from 0.0.0.0/0 over port 80 or we won't be able to serve our customers.

We continue with this design:



THE CUSTOMERS TIME IS VALUABLE, MINE IS PRICELESS

If there's something I'd rather not do is waste my time on waiting, only to find out I've waited for nothing. That's why I've decided to focus on the RDS stack last. It's notorious for taking at least 20 minutes to fire up so it's better to build that stack last (especially when nested). We now focus on the application servers. There are a couple of factors to take in consideration.

The most important is the way the third party wants to deploy new version of Magento and how fast new instances need to be able to launch inside the auto-scaling group.
At the start of this year I've read the book "Infrastructure-as-Code" by O'Reilly. One thing I've learned from this book is that you need to make choices on how you deploy code on a new instance. Baking an AMI with the application already on it severely shortens the time needed to boot and allows the environment to scale fast. It does however involve a tedious process which involves baking a new AMI with every update in AWS Codebuild or, worst case, manually. We don't want this, so I've decided to use a semi-vanilla CentOS AMI built by the Sentia standards.

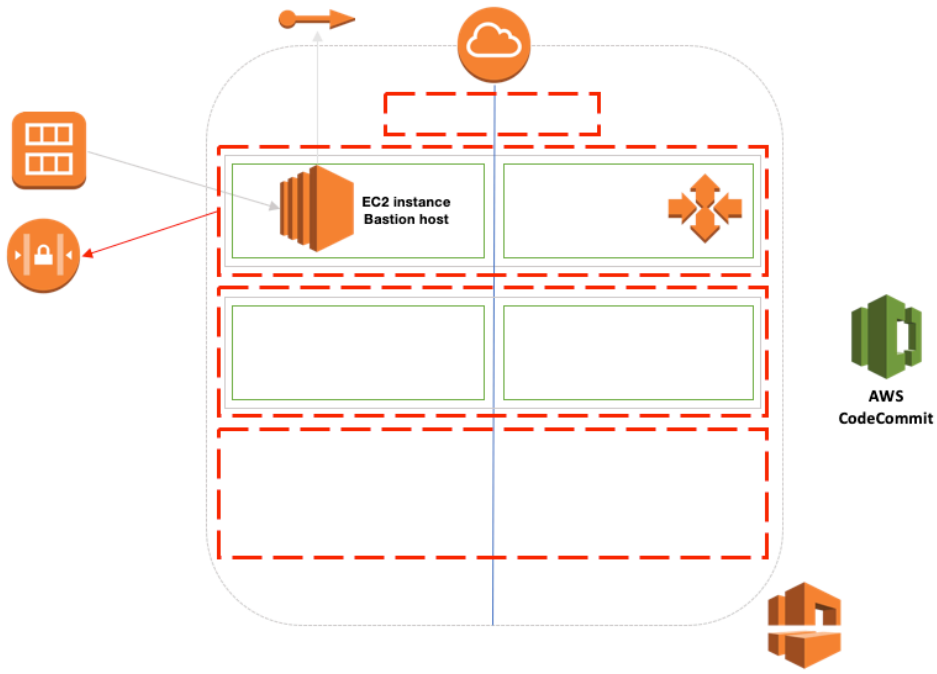
An alternative approach could be using Docker containers. They're lightweight so deploying them is fast (and much faster than launching full VM's). It gives the developers another advantage: environment parity. Deploying a new version of the application is just a matter or rolling out new Docker images. We do however have to keep in mind most Magento developers are nowhere near working with Docker. So, let's stick with classic VM's that connect to a repository and collect the application there. It's not as fast but at least we can integrate nicely into the development tools of the third party.

THE APPLICATION SERVERS

The application code needs to come from somewhere. External repositories come with a risk of not being available, so I've decided to use AWS CodeCommit. It's reliable and we can set repositories to private which reduces the risk of exposing secrets by the third-party developer (we all make mistakes from time to time).

I would like to repeat there should be a conversation with the third-party developer first to make sure we're not building something snowflaky that interferes with the developer's way of working. Let's assume this is how they want us to build this.

We're still working on the skeleton so I'm creating the subnets, routing tables and nACL's to accommodate the EC2 instances. We now have this design:



We have chosen the classic VM deployment method which means we need a NAT gateway because the application servers reside in a private subnet (no routing to the IGW). What is the worst thing that could happen with a NAT gateway? It could be unavailable due to AZ's being unavailable... It's not possible to place a NAT gateway in an auto-scaling group so we need to find another way of ensuring that the application servers can download essential files in case an AZ becomes unavailable (which triggers the launch of more EC2 instances in the other AZ's, so this part is vital!).

Writing this made me reconsider the HA properties of a NAT gateway since it resides in one public subnet only. Obviously, the NAT gateway has built in redundancy. But how does it work and can we depend on it? After reading this piece of documentation I stumbled upon a caveat that can be considered quite valuable.

Note

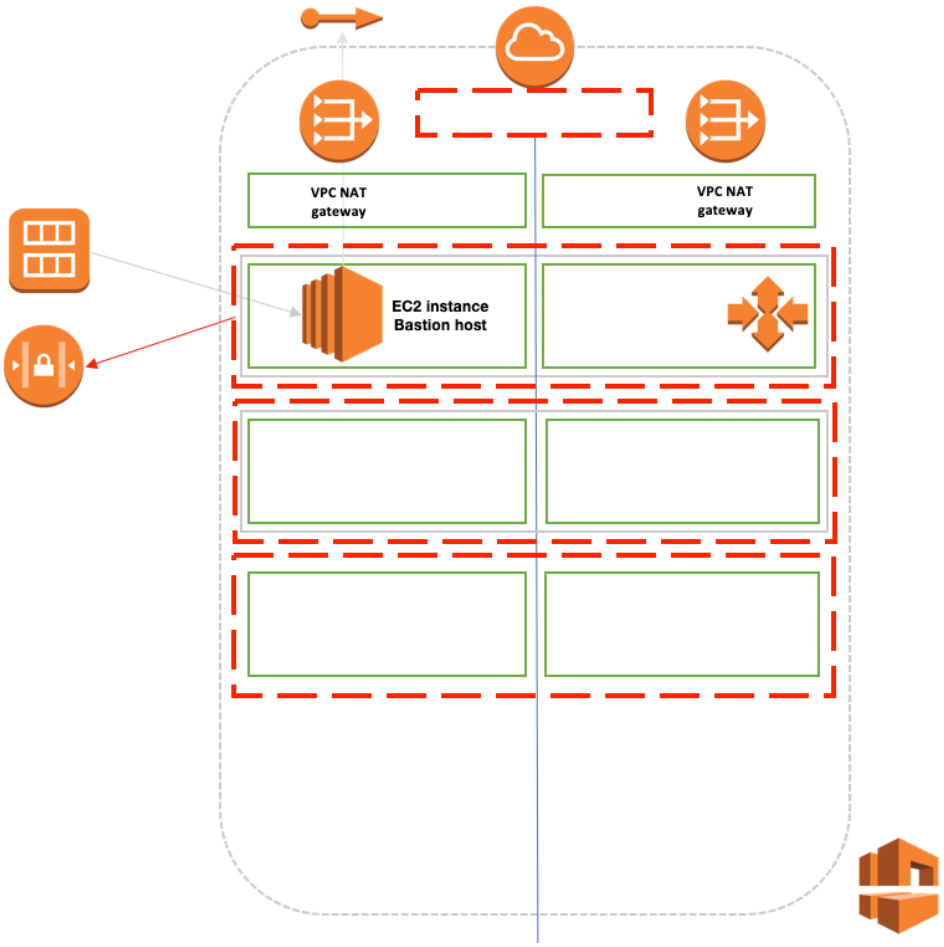
If you have resources in multiple Availability Zones and they share one NAT gateway, in the event that the NAT gateway's Availability Zone is down, resources in the other Availability Zones lose internet access. To create an Availability Zone-independent architecture, create a NAT gateway in each Availability Zone and configure your routing to ensure that resources use the NAT gateway in the same Availability Zone.

https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html

All things considered, I've decided to strongly advise the customer using one gateway per AZ. It would be foolish not to do so.

Creating extra public subnets for the NAT gateway feels like overkill but I feel better creating separate subnets for each function. They're free, it allows for separate nACL's and it's better to not mix different components unless there's a good reason to. I'm supported by the sample architecture on the AWS documentation site.
(https://docs.aws.amazon.com/quickstart/latest/magento/architecture.html).
Sure makes it a lot easier to debug if anything breaks. NAT gateways are not controlled by security groups, I've double check in the aforementioned article.

This is what we've come up with thus far:



We're off to a great start! The observant viewer has probably spotted the extra two subnets in the database group. I've added those so we're good to go and start working on the other layers!

THE APPLICATION STACK

Now it's time for the application stack. Lots of things can break in this stack so we need to keep an open eye for any component that's not at least HA.

First we need to decide on what family of instance and sizing we're going to use, how we want to configure scaling and how we can make this template work in every AWS region. And then there's the deploy method.

FAMILY AND SIZING OF EC2

Choosing the right family and sizing is key to getting good performance. Magento 2 can be a very compute intensive application. At the same time, we don't want to use instances that are too big as it's preferable to spread the load over multiple smaller instances then e.g. just two big ones. If an instance breaks it's not a big of a deal if there's six others to take over. If it's just one and you need to wait for the new node(s) to boot it can mean the difference between catastrophe and a successful tv-campaign.

The experts of MGT use c5.xlarge EC2 instances. In Frankfurt and at the time of writing these instances would cost 0.1940 USD per hour (on-demand). Let's investigate if we can get a better deal than this and investigate what is seen as a best practice.

The Magento 2 documentation states every server needs at least 2GB of RAM.
https://devdocs.magento.com/guides/v2.2/install-gde/system-requirements-tech.html
In general, I've learned it's the application servers that are CPU hungry and the database servers that are memory needy. So it makes sense to use different families for RDS and EC2.

Looking at the plans of Hypernode (https://www.hypernode.com/pricing/) you would presume 4 cores and 8GB of RAM are the bare minimum but we need to keep in mind that these are single VM's that need to accommodate both MySQL and Nginx.

After doing some Googling I've found these pieces of advice:

1. Invest in fast hardware; don't just look at the number of cores and gigs of RAM. https://community.magento.com/t5/Magento-2-x-Hosting-Performance/Ideal-EC2-Instance-type-for-magento-2-2-3/td-p/91990
2. Benchmark different instance families and sizes before going into production. https://docs.aws.amazon.com/quickstart/latest/magento/design.html

Side note: the second page is part of an official AWS QuickStart and does confirm my first thoughts on what services to use. Great!

In general, it is recommended to use the C family because it's equipped with more compute. Looking at this page gives me the idea to go for C5's but I notice the I/O is limited to 10 Gbps. https://aws.amazon.com/ec2/instance-types/

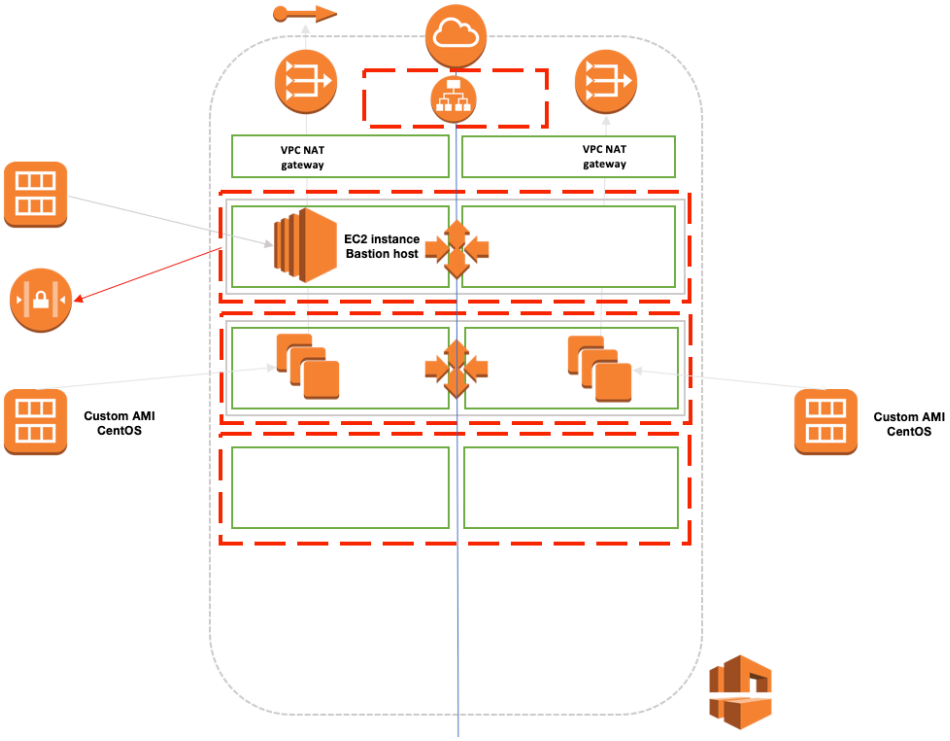
The C4 family gives a bit less RAM and is a bit more expensive but does offer "high" I/O, which I know can make the difference in Magento. Byte seems to be going for the C4's as well as their posted specs include 7.5GB RAM, which isn't available in the C5 family. They certainly know what they're doing so I'm going for the C4.xlarge for now. I'll ask one of my old colleagues if I'm doing the right thing here.

The difference between the C4.xlarge and C4.2xlarge is the price doubling. Looking in the AWS calculator it tells me I could save 53% on either family when using reserved instances. Something to consider. On-demand does have its pros and cons.

AUTO-SCALING, HOW TO SCALE

The promise of auto-scaling is that you can quickly and automatically react when a VM gets corrupted or there's a sudden spike in visitors. I plan on leveraging target tracking with the ALB (a.k.a. ELBv2) and tweak the settings after the migration.
https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-target-tracking.html
Magento is CPU hungry so cpu-utilization seems like a good start. We want to use 1-minute metrics to ensure a fast response. Don't forget the AMI has to download the whole application so every second counts!

The architecture now looks like this:



HAVEN'T WE FORGOTTEN SOMETHING?

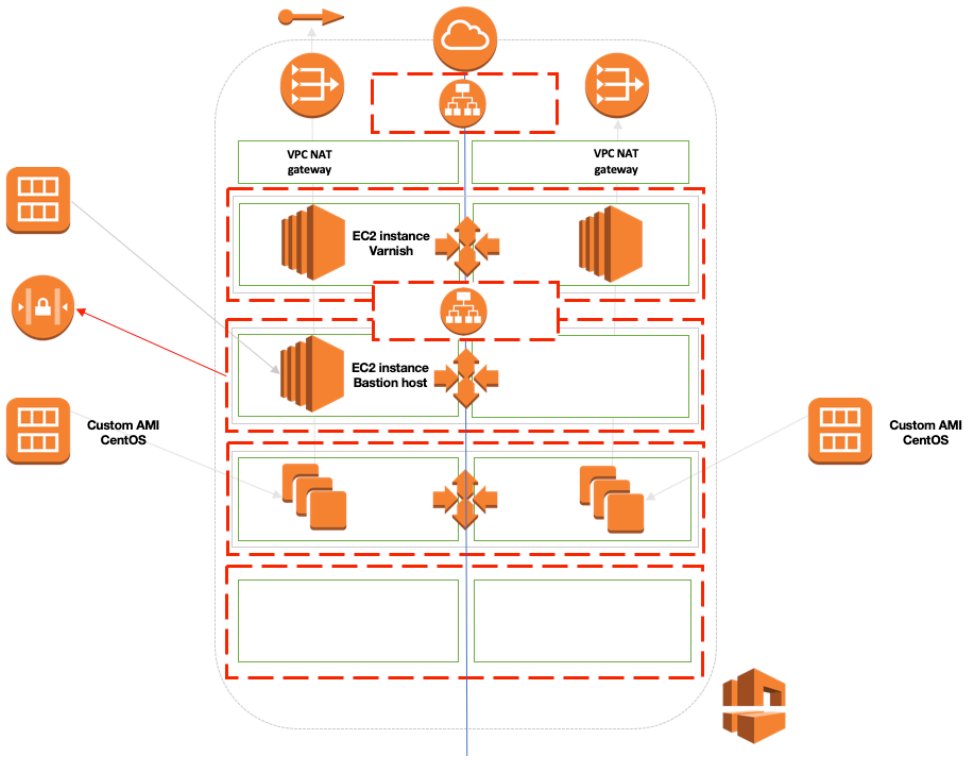
Whilst working on the application layer, I suddenly realized I haven't created subnets for the Varnish instances. A fast Magento 2 website needs to leverage Varnish, a Full Page Cache. Let's correct this.

I've once read it's bad practice to install Varnish on the application servers, you want to have at least two separate instances for HA purposes and you need an internal load balancer. Varnish requires the addresses of all back-end servers to be defined in the config file. This poses a problem since we don't know for sure what instances exist in an auto-scaling group. Every instance is ephemeral. Just implementing a ELB isn't going to change this because Varnish requires a static ip and the ELB doesn't have one. Fortunately, our friends at MGT-commerce have solved this by implementing a PHP script using the AWS SDK for PHP.
https://www.mgt-commerce.com/blog/magento-auto-scaling-with-varnish-on-amazon-aws/

Building this script will cost time. Luckily my colleagues in Nieuwegein were confronted with the same issue and have solved this for me. This is the script that works like "Haarlemmerolie": http://cloudreach.blogspot.com/2013/01/varnish-and-autoscaling-love-story.html

The article mentions it can be run as a cron. Let's check with my peers in Nieuwegein tomorrow.

The design now looks as follows:



ELASTICACHE

Magento uses Redis for sessions, backend and full-page caching. Elasticache is an AWS managed service. After reading the documentation on:
https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/GettingStarted.CreateCluster.html
it states Elasticache lives in at least one subnet per AZ and requires a security group. Let's add those to the design now and compliment them with an Elasticache Redis cluster.

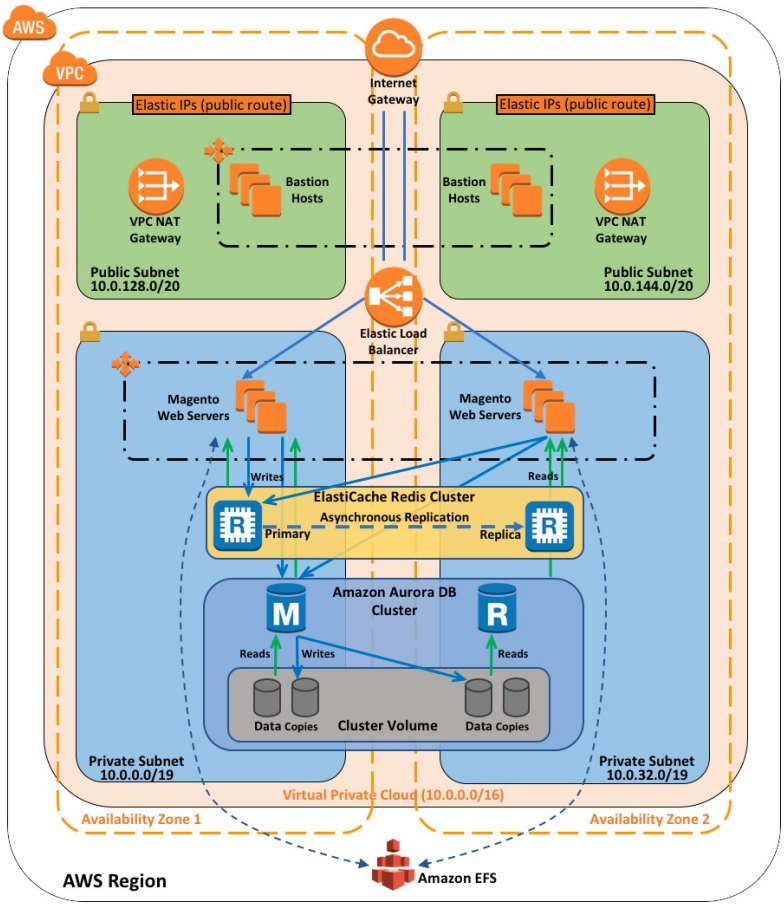
I have never worked with Elasticache, so when building these templates, I'll be sure to first read the documentation and the Magento requirements.

RDS STACK

Now it is time for the cherry on the cake: the RDS cluster. In the Sentia Academy I've found that setting up an RDS cluster takes a long time but AWS did a great job making it easy.

In one of the previous projects I've worked on my colleague used AWS Aurora as a database engine. Apparently, it's much faster than MySQL and can be leveraged as a drop-in replacement. It doesn't require you to rewrite your application. The quickstart here:
https://aws.amazon.com/blogs/database/accelerate-magento-content-services-deployment-on-amazon-aurora-with-aws-quick-start/
mentions it as well.

This diagram shows the standard implementation of RDS and Elasticache.



SENTIA

There is a couple of things that stand out in this diagram. One of them is the use of EFS and the other is there are only four subnets in use. EFS is notorious for performance issues in combination with Magento 2. I've left it out for that reason and decided to upload all static content to S3 instead. This does take some adjustment in Magento but has proven to be not too labor-intensive. We can always use NFS or EFS as a backup plan if changing to S3 doesn't work out.

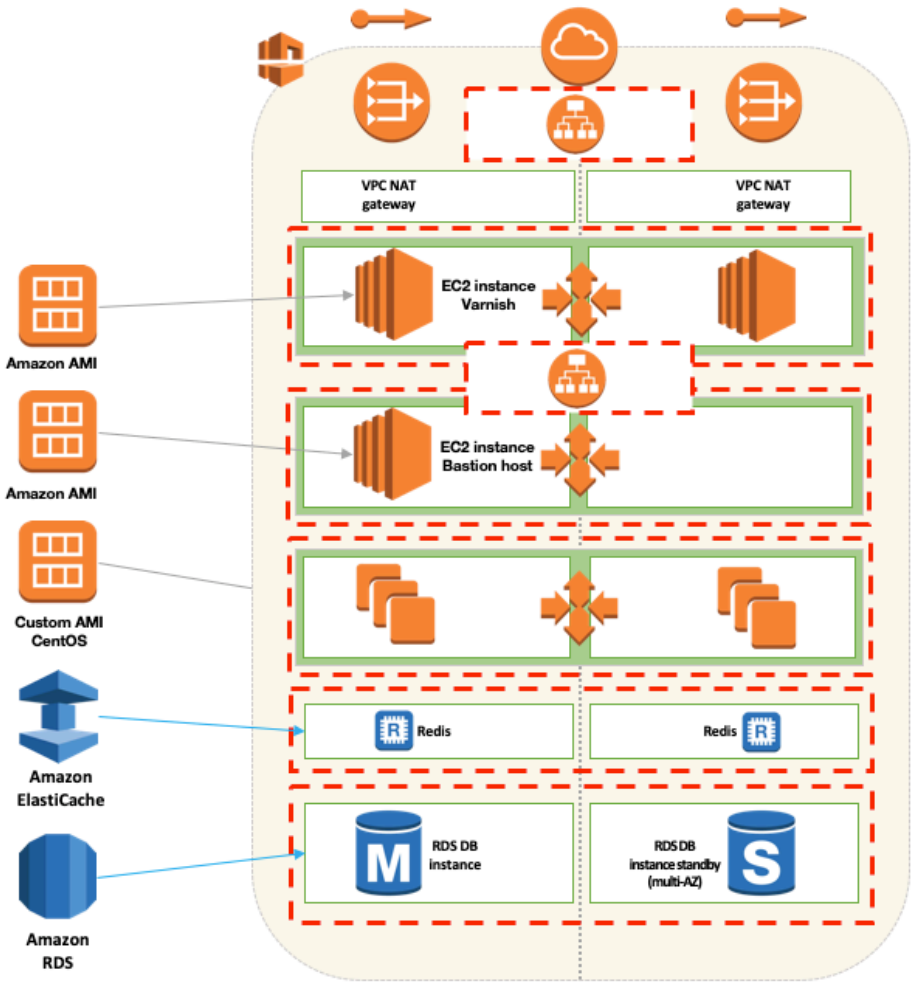
The second thing that catches the eye is the use of only four subnets. I assume this is done to keep the setup basic and simple. In my architecture I've added many subnets. I've done this deliberately as they're free and allow for the use of nACL's, creating as many barriers as possible for visitors with nefarious intents.

Ok, back to the RDS stack. We haven't discussed this before but I've used nested stacks for the Cloudformation templates. Nested stacks have some great advantages. Templates can easily be re-used many times over. If our customer decides to build an extra testing environment we can create it with a flick of the switch. Same goes for a acceptance environment or a temporary environment for the developers to use as a sandbox.

Then there's the RDS stack. In my opinion this stack shouldn't be nested along the other stacks. The data inside the database is the most valuable part of the entire environment. It should be left on its own and safely guarded from the other stacks. If anything goes wrong, we can at least rebuild the other stacks and re-attach them to the RDS stack.

This does come with its pros and cons. One of them is that we now need to export the values which makes them available for all other stacks in the same account. This can be error prone if the full DTAP is in the same account. Therefor I would strongly recommend creating a separate account for the production environment, if possible for all environments. This way the so-called blast radius is limited to what happens within this one account, and it solves our problem with the exported values.

So, let's see what the design looks like now.



We are getting pretty close to a full design but there are still some components missing on the right side of our diagram: the deployment services.

DEPLOYMENT SERVICES

Deploying an application can be done in many ways. Magento 2 comes with its own particulars. We need to think about flushing the Varnish cache after every deploy and we need to decide how we want to deploy. Canary? Blue-green? Or a rolling deployment? There are many options and we luckily don't have to decide yet because we use AWS Codedeploy. AWS Codedeploy comes with an agent that needs to be installed on the EC2 instance when booted. Per deployment group you can choose the deployment method. It integrates nicely into AWS Codepipeline, a service I've learned to use in the Sentia Academy as well.

The application is pushed by the third-party developer via git to AWS CodeCommit. Preferably the master branch is protected and contributions can only be made through pull requests. For each environment a separate CodePipeline is deployed, each listening to a different branch of the same repository. If a new version of the code is pushed to the repository, CodePipeline will execute consecutive steps. CodeDeploy will be the last step, deploying the new application to the EC2 instances. CodeBuild is a vital part of this pipeline. In CodeBuild we collect the secrets stored in SSM Parameter Store and create the build that is deployable as a whole.

This part of our project can be tailor-made to fit the needs of the customer or the third-party developers. It allows is to create a pipeline that includes integration tests, performance regression tests and more.

CDN AND STATIC CONTENT

One of the requirements stated the need for speed. Implementing a CDN (like AWS Cloudfront) can have a very positive impact on load times and as a side-effect obfuscate the ip-address of your production environment. This makes it just a bit harder for the bad guys to DDoS your website. Implementing Cloudfront doesn't have to be hard but it does help if your static content is already in an S3 bucket. Magento has a vibrant marketplace and a large contributing community offering lots of solutions for S3 storage. I've found this thread with some interesting links. https://magento.stackexchange.com/questions/462/magento-media-assets-in-amazon-s3

To make sure I'm not doing something stupid here I will consult my befriended engineer and would talk to the third-party developer first. Big chance they have a ready-made time-proven extension to use S3.

In the Magento config files you can change the prefix for all media content. After creating a Cloudfront distribution, I will use the CloudFront DNS endpoint for a subdomain called https://static.website.com/

LOGS AND BACKUP

Logging is a key component in this setup. EC2 instances are to be regarded ephemeral so all logging needs to be stored in one highly-available data store. For this we use CloudWatch. The CloudWatch agent will run on the application servers and send the system logs as well as the Nginx logs to CloudWatch. The logs can then be stored in an S3 bucket which resided in a separate account, to ensure the integrity of the log files.

Backups of the database are performed daily and can be retained between 0 and 35 days. In that time period daily snapshots are complemented with transaction logs, allowing us to restore the database to the second within the given retention period. After that period, we will automatically store the daily backups on S3 and install a lifecycle policy. The retention periods are set to the requirements of the customer.

The S3 bucket is redundant within the region and offers 9 nines durability. If required we can configure versioning and cross-region replication.

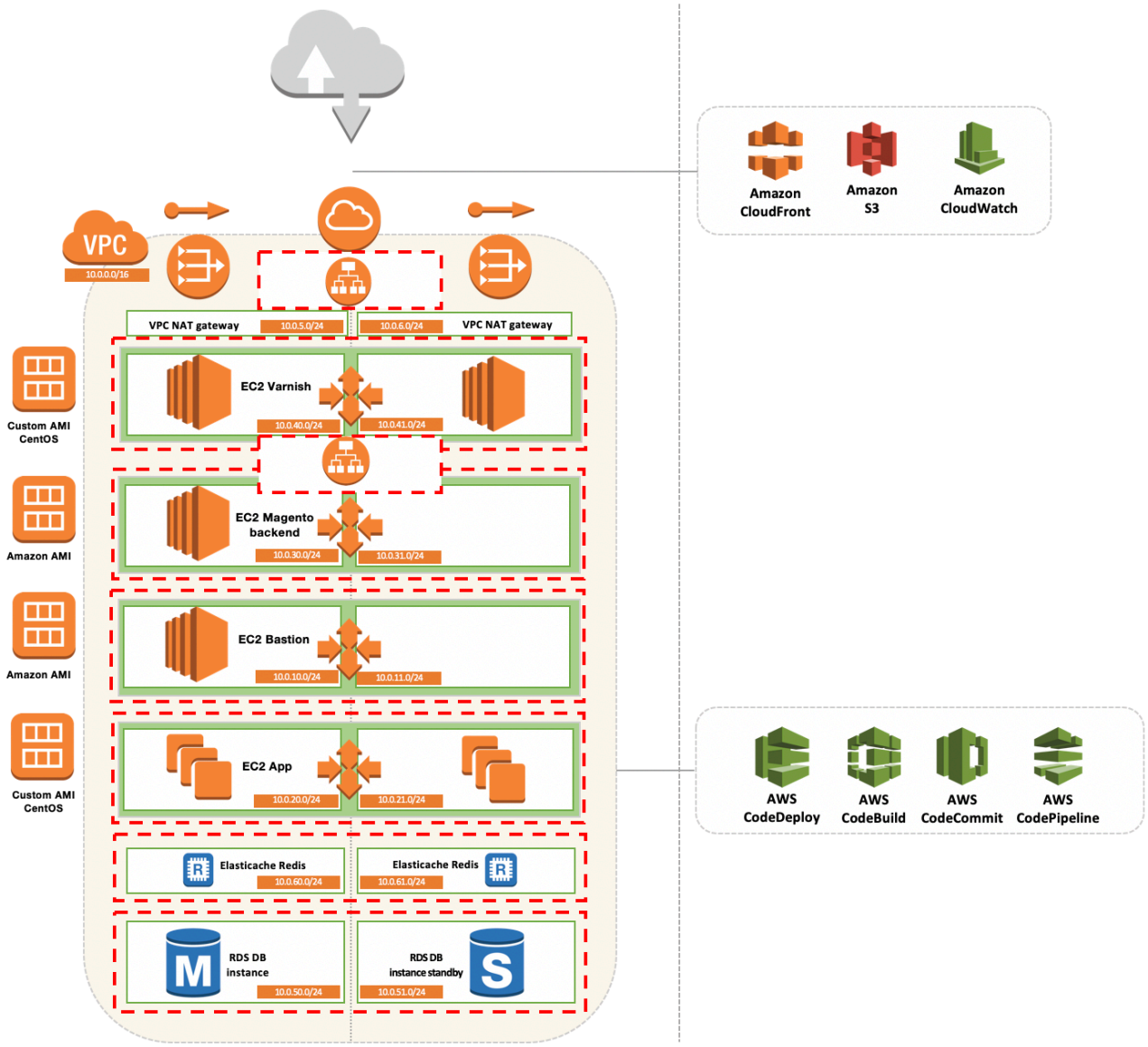
BACKEND SERVER

A great piece of advice from my friend at Byte is to integrate a separate backend server. When this website grows, a lot more employees will rely on the backend. To prevent the frontend from slowing down the backend and vice versa, a separate EC2 instance is deployed using the exact same AMI, codebase and deployment method. Only difference is the ELB routing backend traffic directly to this instance, skipping the Varnish instances and internal load balancer.

It might seem like overkill now, but it can help you dodge a lot of bullets when traffic peaks and your business grows later.

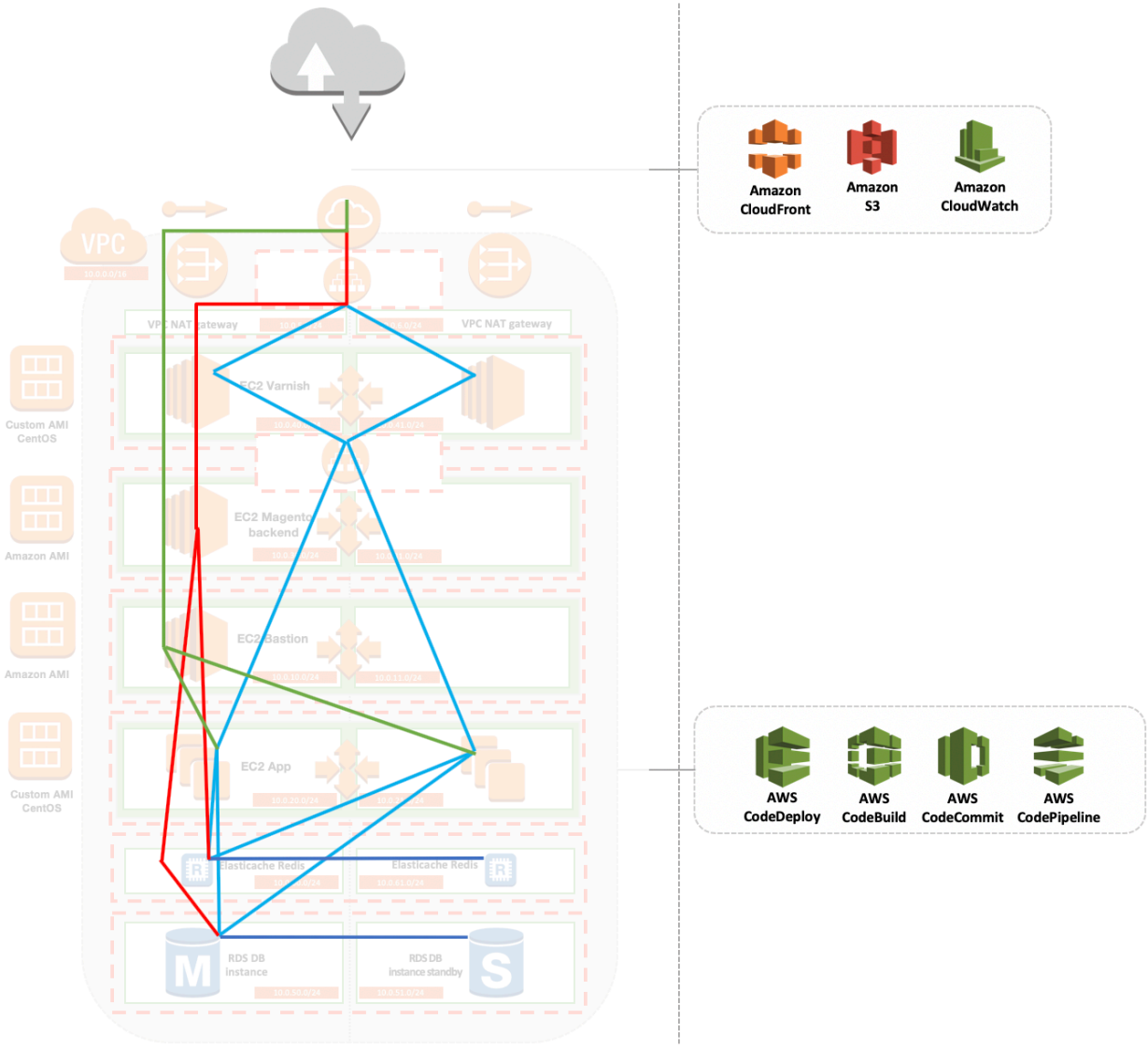
FINAL DESIGN FOR QUESTION 1

In the final design I've inserted the ip ranges, deployment services, backend server and tweaked the design a bit.



OVERLAY TO SHOW TRAFFIC

This design doesn't fully show how all parts are connected, although I've stacked the parts in a logical way. Therefor I've created a separate overlay that shows how traffic is routed. I've excluded traffic between instances and the NAT gateway to prevent clutter.



CONSIDERATION

In real life I would present this design to the customer and the third-party developer. If all parties consent, and only then, we start building the environment. I need to make the templates as re-usable as possible, so we only have to maintain one template. It assists in maximizing environment parity as well which will save the customer and Sentia a lot of unplanned work and time on debugging.

CONCLUDING QUESTION 1

In this part of the assessment I've come up with a design based on AWS and Magento bes practices. This design allows for the customer to grow without having to implement major changes. It's fast for SEO purposes by using Varnish and Cloudfront. I've looked into picking the right family of EC2 instances and choose the right sizing. The environment is highly-available to prevent lost revenue and SEO penalties.

There are some choices that have a direct impact on the cost. For example, the double NAT gateway, the time it takes to implement storing static content in S3, and the choice for a rather traditional way of deploying (instead of e.g. containers). We could make this design more cost-effective by using reserved instances and tune the sizing.

Obviously, in the first few weeks we'd rather use too many resources and tune different components down, than starting off with downtime. I've heard good stories about Blackfire so I would definitely use that to further streamline the design.

The only part that feels a bit sketchy/hacky is the integration between Varnish and the ELB. I've talked to one of the Magento guru's, Sander Mangel, and he told me he has been using this method for years without if ever breaking down. I would definitely add some custom monitoring here to make sure we know when it breaks, and it doesn't catch us by surprise.

QUESTION 2

Product reviews are integrated in Magento 2.
https://docs.magento.com/m2/ce/user_guide/marketing/product-reviews.html

I don't see how this could affect the architecture besides being more strenuous on the RDS cluster. If the built-in features don't suffice the customer could switch to a third-party built extension. In general, most third-party developers have their own way suite of extensions. It is wise to review the code and monitor the impact of a third-party extension with tools like New Relic or Blackfire. Apart from that I don't see any need to alter the architecture.

QUESTION 3

Centralized storage with built-in redundancy...S3 would be my choice of weapon here. We could combine it with a Lambda function that triggers every time a new file is uploaded, creating a thumbnail with a predefined object name and in a dedicated bucket.

A quick Google search gave me these results:
https://concrete5.co.jp/blog/creating-video-thumbnails-aws-lambda-your-s3-bucket
https://github.com/deek87/lamba-thumbnailer
https://github.com/shufo/python-lambda-image

This looks promising but let's take a step back. What do we want to accomplish here and how does it fit in the bigger picture? Wouldn't it be great if we could use an AWS service which possibly offers more functionality, instead of a custom script which introduces all kinds of dependencies, etcetera?

After some more digging I've found a blogpost from our friends at A Cloud Guru. They have tons of video's and even more thumbnails.
https://read.acloud.guru/easy-video-transcoding-in-aws-7a0abaaab7b8

Now here is a thought: could our customer for example expand business over the borders or have some other future requirements? How about different format, subtitles, resolutions or target devices? It doesn't make sense to just thumbnail a video of 2GB in some exotic format. And how about watermarking? Could our customer ask us to develop a whole set of features next?

AWS Elastic Transcoder can do all of the above for us, and more. We can use alerts to trigger SNS or Lambda functions and more... (hint, it has something to do with the next question).

Before we go berserk and start building the templates...let's look at the limitations of AWS Elastic Transcoder first.
https://docs.aws.amazon.com/elastictranscoder/latest/developerguide/limits.html

The biggest limitation is the maximum of 4 pipelines per AWS account. If we'd put all environments in one account this would be a blocking limitation. But we've decided to follow the best practice of placing all environments in separate AWS accounts, so we should be ok here. There is one more thing to keep in mind: this service is available in eu-west-1 but not in eu-west-2. Earlier on we've decided to stick with eu-west-1 but we need to be aware this constricts our ability to easily switch over to another European region.

Now we have to make decision that defines the way we build this solution: Elastic Transcoder is, as of yet, not supported by CloudFormation. I've looked in this list:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html

We now have to choose between two evils. We either settle with that we can't deploy this part of the environment from CloudFormation templates, or we choose scripting everything in Lambda functions which has a lot of downsides on itself. If we choose to script everything ourselves, there is no way back. If we decide to create the Elastic Transcoder manually, we at least have the hope this will be available in CloudFormation at some point in the near future. Let's stick with creating the Pipeline manually. If it breaks new video reviews will be shut down for some time.

After taking a closer look at the assignment and the scope, I have to conclude the guys from A Cloud Guru got me overthinking this. I don't want to use ET because it doesn't work well with CFN and it does much more than just creating thumbnails. We can still offer it to the customer but let's tone the solution down to just a bucket with thumbnails.

INTEGRATING OUR SOLUTION INTO MAGENTO

This is the part where it can get hairy. How do we integrate this functionality into Magento? We could build this entirely in AWS, but we still rely on Magento to show the correct video in combination with the right product. Of course, we could build a PHP script on the product page

which checks an S3 bucket for an object with a certain prefix and skip if it doesn't find it. We could use the AWS PHP SDK for that. But let's take a step back.

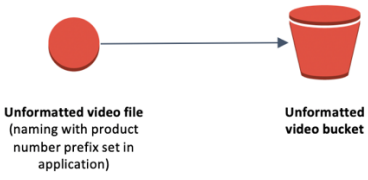
Aren't we building something knowing there's a better solution? SEO is important for this customer so why do we rob the website of visitors from video platforms, like YouTube and Vimeo? If we store the video's in S3 we're hiding them from a bigger audience. I understand the assignment, but I do want to put this out there to show I always try to understand the bigger picture before I write any code.

THE SOLUTION

After a couple of cups of coffee, I've decided to keep the solution simple.

All videos are uploaded to an S3 bucket. How is something that's out of scope for this assignment (probably through the use of the AWS PHP SDK). It will require at least some customization, but it is clear we need to have at least some reference to the products in the object name. Otherwise it will be hard for us to link the video to the correct product, or we would have to use some sort of key value store to keep track of all the files. This means more moving parts that can break and add up to the cost.

Product variations are out of scope as well. In Magento one can find many different kinds of products (https://docs.magento.com/m1/ce/user_guide/catalog/product-types.html). We assume this assignment is about simple products.



We start with one S3 bucket. This bucket, as a good practice, has restricted access. The maximum size per object is restricted in the upload form. To make sure old files are deleted after an x amount of days we set a lifecycle policy on the first bucket.

DELIVERY METHOD

We have a way to ingress videos, process them and deliver output to two S3 buckets. But we need a way to tell Magento if there is a video available and come up with a way to display it on the product page.

Good old Google showed me to this page:
https://docs.aws.amazon.com/aws-sdk-php/v3/api/api-s3-2006-03-01.html
And this one:
https://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.S3.S3Client.html

There seem to be a couple of ways to do this. The easiest (and dirtiest) way is to use the AWS PHP SDK and check if the object in the bucket with the product number in its key exists. Enough about the delivery methods. They're the most simple part of this whole setup.

FIRST STEPS

To start off I've created a CNF templates which spawns three buckets. An origin bucket for the user input, a bucket (90 days expiration rule, I'll come to that later) for all videos up for approval and a bucket for approved videos (yes, I've read the next question and don't want to work on this twice).

Let's look for a good candidate in our Lambda function to thumbnail the videos. A quick Google showed me the way to this GitHub repo:
https://github.com/shufo/python-lambda-image

As this is the third lambda function I've ever written, let's look at the prerequisites. For starters importing any library doesn't work. I've found out you need to create a deployment package first.
https://docs.aws.amazon.com/lambda/latest/dg/lambda-python-how-to-create-deployment-package.html

After writing a working Lambda function that uses ffmpeg I've concluded I was wrong to use just Lambda. The deployment package has grown to over 100MB and I didn't take the maximum of 15 minutes of Lambda in account. Rookie mistake.

This is what I've done so far (rewritten so I can run it local in a Docker container, instead of on Lambda):

```
import boto3
import os, sys
import time
import ffmpeg
from datetime import date

s3 = boto3.client('s3')

def lambda_handler():
#    origin_bucket = event['Records'][0]['s3']['bucket']['name']
#    key = event['Records'][0]['s3']['object']['key']
    origin_bucket = 'video-review-origin'
    key = 'small.mp4'
#   !line below needs to be rewritten when used in lambda!
    hierarchical_key = 'video/small.mp4'
    pn = key.rsplit('.', 1)
    pn = pn[0]
    stamp = time.strftime("%Y%m%d%s")
#   combination of stamp and productnumber is unique
    desired_thumb_name = (str(pn) + "-thumbnail-" + str(stamp) +
".png")
#   s3 allows for hierarchy in object by adding a slash
    hierarchical_thumb_name = ("thumbnails/" + str(pn) + "-thumbnail-
" + str(stamp) + ".png")
    source = {"Bucket":origin_bucket , "Key":key}
    target_bucket = 'non-approved-files'
    thumb_bucket = 'non-approved-files'
#   waiter = s3.get_waiter('object_exists')
#   waiter.wait(Bucket=origin_bucket, Key=key)
#   !above needs to be changed back when using as lambda!

#   copy to non-approved-files bucket and download file from s3
    s3.copy_object(Bucket=target_bucket, Key=hierarchical_key,
CopySource=source)
    print("file copied")
    s3.download_file(Bucket=origin_bucket, Key=key, Filename=key)
    print("file downloaded")
#   let the magic begin, take a screenshot
    os.system("ffmpeg -i %s -ss 00:00:4.435 -vframes 1 %s-thumbnail-
%s.png" % (key, pn, stamp))
#   upload the thumbnail to the non-approved-files bucket but in a
hierachy
```

```
    s3.upload_file(desired_thumb_name, thumb_bucket,
hierarchical_thumb_name)

lambda_handler()
```

The above script works and delivers a thumbnail in the desired bucket. But I'm not happy since I've spent more than four hours finding out I was chasing the proverbially red herring.

NAILS AND FAILS

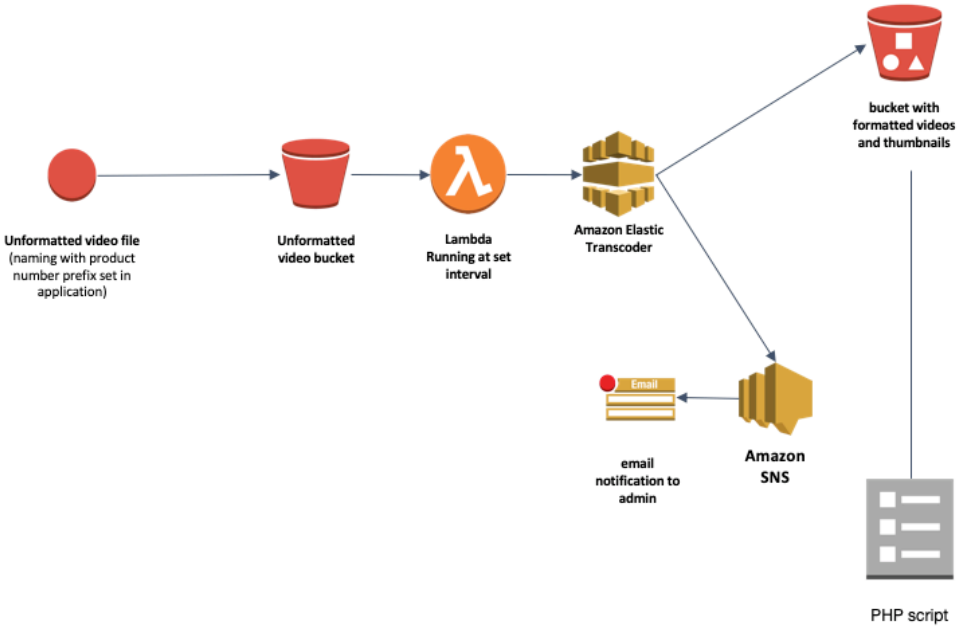
So, it's back to the drawing board. Let's look for an example Lambda function which creates an AWS ET job. I've found some interesting information in the boto3 documentation: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/elastictranscoder.html#ElasticTranscoder.Client.create_pipeline

Apparently, you can create ET pipelines through boto3, which kind of solves our problem of ET not being available in CFN. Five minutes later I have a pipeline, so I guess this works! Great learnings! Let's not only create a pipeline, but a preset and job as well.

Thirty minutes later I'm stonewalled again. In CFN you can reference to other resources. In Lambda you have to rely on list functions. Happy to say there is one and I can return the Id of the pipeline I just created and use it for the job creation function. Same goes for the preset we've just created. It will be a lot more work than with CFN integrated services though.

After getting it to work locally, and spending many hours debugging, I've tested it on Lambda and now I can start integrating it into my CFN template and finish this assignment up.
I've learned many things from this assignment. You have to work around the limitations of CFN and AWS, and MacGyver to a solution that's fault-proof. There's a lot of things that can break and you need to take them all in account.

The lambda function and CFN template are in the appendix folder. Below is the overview of our working solution.



QUESTION 4

Creating an approval process should be fairly straightforward. We will maintain the architecture of the previous diagram but add an extra bucket for the approved videos complimented with an additional Lambda function which moves the videos between buckets.

There is one hurdle to take though. How do we interact with the admin and display videos pending approval? The easiest way would be to create a user that is only allowed to view the content of the approval bucket and allow it to copy/move the files to the approved bucket.

We could create this user (or role) in the Cloudformation template and that would be the end of it. Mission completed. Or is it?

THE GOOD, THE BAD AND THE UGLY

I have to admit this solution isn't very pretty. It forces the admin to leave his Magento backend, login to the AWS admin console and move the video files by hand. Apart from being ugly, it's quite fault-prone. If the admin changes the filename, the video review will fall through the

cracks and disappear from the website. Worst case it will show up on another product page. Or the admin could delete it. So, what's the alternative?

The simple alternatives are not very abundant. I could go through the trouble of creating an API Gateway and linking it to a Lambda function. The Lambda function would move the file between buckets and will be triggered if the user sends a request to the API Gateway endpoint. That could be a request in the form of a link in an e-mail that is triggered by the S3 bucket. That's a lot of coding and comes with its own pros and cons. We shouldn't over-engineer this if there are only 5 video reviews a month. But what if there are 5 a day, or 5 per hour? That's almost like a small e-mail bomb going off. Let us think of something more graceful.

I've tried to look for a service in AWS that takes an S3 object as a source, asks for approval and can then trigger an action to move or copy the same object.
Some Google searches later I've found this tutorial:
https://dzone.com/articles/continuous-delivery-to-s3-via-codepipeline-and-cod
https://github.com/stelligent/devops-essentials/blob/master/samples/static/pipeline.yml

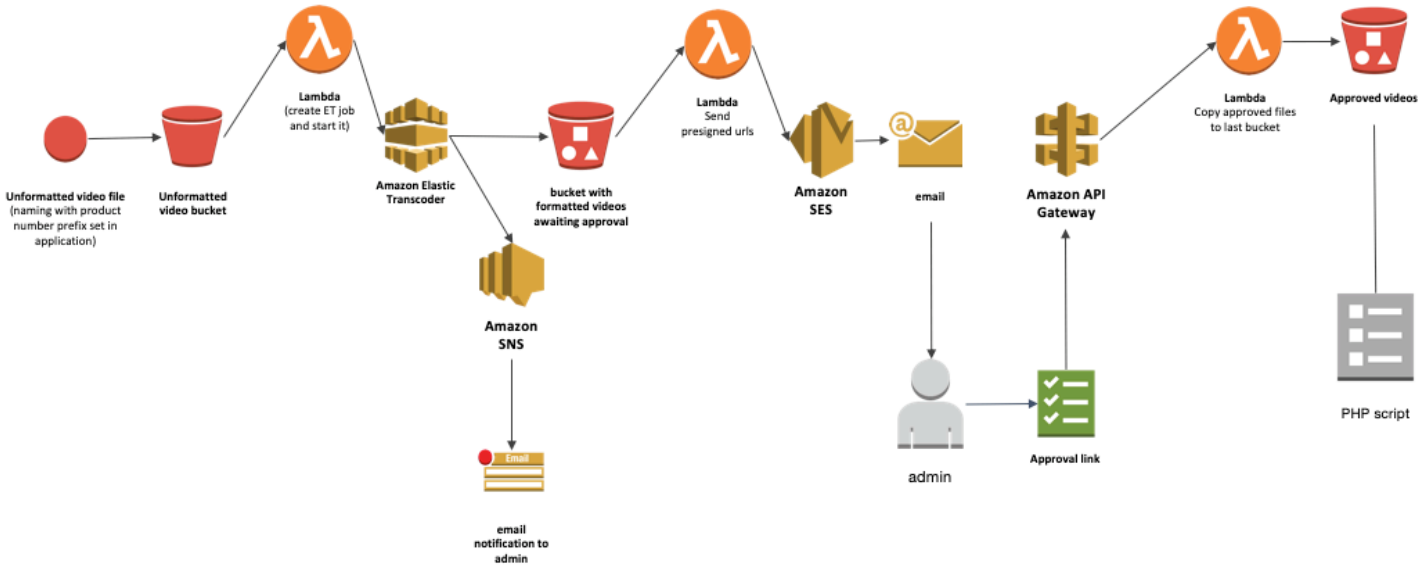
I initially thought this could be a dirty hack to get an approval flow working. But I was mistaken. CodePipeline only takes zipped files as a source, and a workaround would take me even longer to build. Too bad, this is a dead end.

After some investigation I've found out there are not that many ways to interact with a user. Slack was an option but would be a bit too much for this assignment, so I've decided to interact with the user by e-mail (AWS SES) and stick with the first mentioned solution.

The CFN template generates three buckets. One for the uploads, one for the processed videos and thumbnails, and one bucket for the approved videos and thumbnails. Every hour a lambda function is executed which checks for new files, send them to AWS Elastic Transcoder, generates a pre-signed URL and then e-mails the admin with a message stating there is a new video for approval. In this e-mail the admin can click two links. One link serves the pre-signed link to the object in the S3 bucket. I've used pre-signed URLS's because it's safer (the videos are not approved yet, right?). The other link routes to an API Gateway which in turn fires up another Lambda. This Lambda copies the approved video and thumbnail to the third bucket. The third bucket can be used as a source for the product page.

Cloudformation and Lambda are very well integrated. Because of my lack of experience, I sometimes had to find a way to solve issues that most probably could have be solved easily. A good example is the api-id which I had hard time to look up using boto3. In the end I've stored it in the SSM parameter store from within the CFN template. The Lambda function looks it up from AWS SSM parameter store. I'm sure there are better ways to do this.

This is the final design:



The whole architecture has been thoroughly tested. Only weak points are SES and the delivery of the e-mails. We could complement this architecture with lots of monitoring to make sure we know when something fails.

LAST WORDS

I've had a great time working on this assessment and I'm looking forward to more riddles to solve. Thank you for giving me the opportunity to work on this assessment and I'm looking forward to hearing your feedback!

Kind regards,

Sjoerd Santema