**DevOps Project**

**(Techgrounds/Sentia)**

**TEAM: THOMAS**

**GOAL:**

**Designing a solution and delivering an environment in the public cloud using Infrastructure as Code (IaC).**

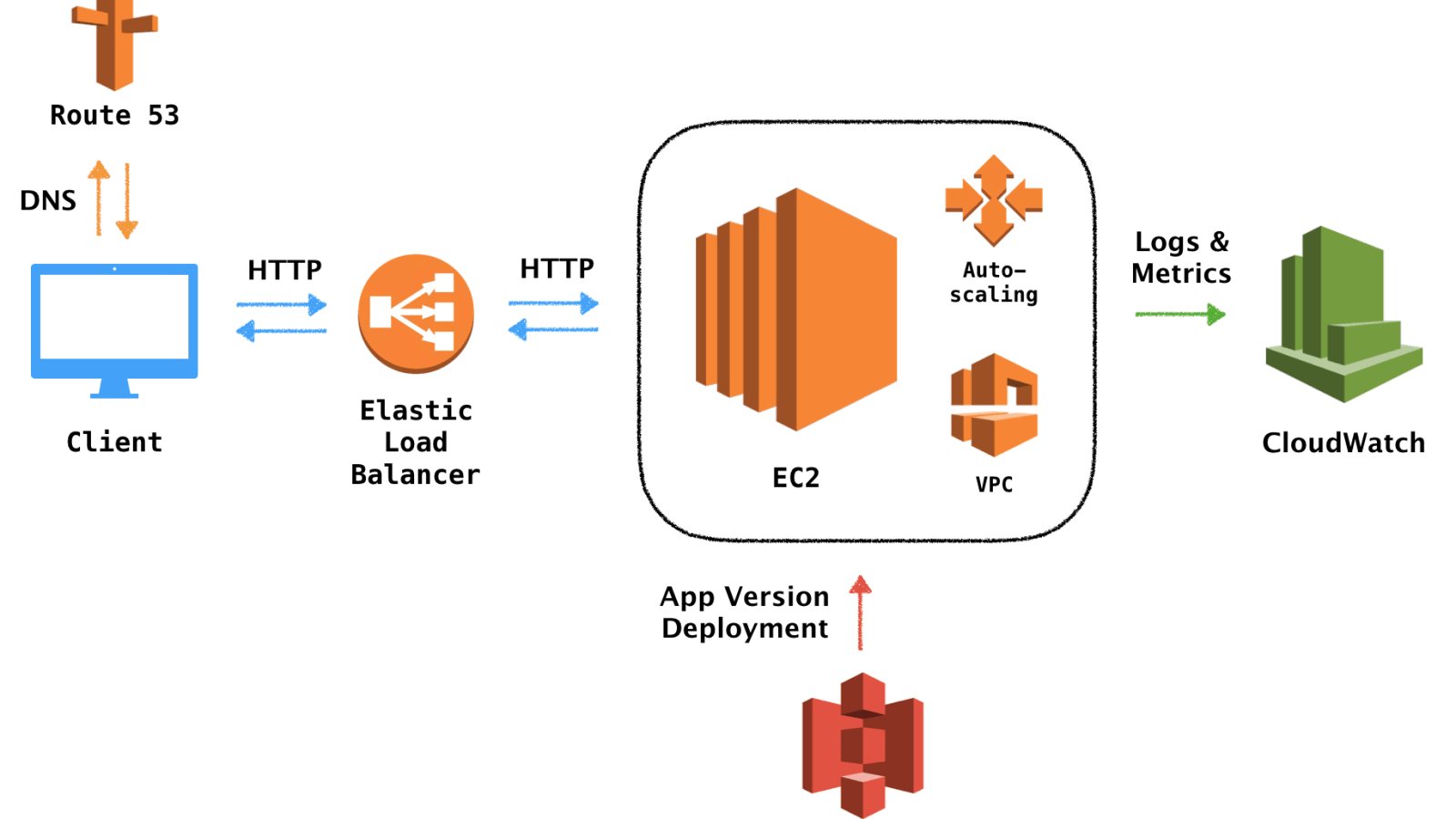
**Purpose:**

**Transformation and migration to the Public Cloud**

**ASSIGNMENT:**

**The assignment is posted in the link below:**

**https://github.com/sentialabs/public-cloud-recruitment/blob/master/ASSIGNEMENT.md**

****

**REQUIREMENTS:**

Design solution

(must) be scalable and flexible.

(must) utilize managed services as much as possible.

(nice to have) be modernized during this migration in terms of infrastructure technologies used.

**ASSIGNMENT ANALYSIS:**

|  |  |
| --- | --- |
| **Existing design choices** | **Replacement design choices** |
| 1.NGINX reverse proxy  2.MongoDB  3.Node.Js applicatie  4.Document storage  5.Cron server jobs  6.ElasticSearch Cluster  7.Dashboard  8.Test, Development and Production Environments  9.Application and infrastructure logs | Elastic load balancer  Document DB  EC2 instances or containers  S3  Cloud watch EventBridge with Lambda functions  DynamoDb  AWS Quicksight  Elastic Beanstalk  S3 |

**CLOUD**

* When compared with Azure and Google Cloud, AWS Cloud is the best Cloud choice and has been truly successful in convincing more traditional businesses to move to the cloud.
* Based on the requirements of this project, AWS is chosen because of its fully managed and modernized services with scalability and high availability.

**INFRASTRUCTURE AND RESOURCES**

**CloudFormation**

CloudFormation is an infrastructure automation platform for [AWS](https://www.contino.io/insights/whos-using-aws) that deploys AWS resources at a scale much faster, more efficient, and more secure.

**Assumptions:**

* AWS CDK expands the number of resources that developers can manipulate through a code base, it offers more functionality than limited tools such as CloudFormation and Hashicorp TerraForm. The AWS CDK not only streamlines the provisioning process, it also simplifies verification and review.
* Terraform might be a better fit if your infrastructure relies on many third-party resources.
* Cloud formation speeds up our process when compared to CDK and terraform as we need to spend less time on code and more time on the application.
* AWS's native IaC platform, CloudFormation offers the deepest level of integration with the AWS cloud, including features like Designer, which lets you create and modify CloudFormation templates directly on the AWS website.
* CloudFormation also provides a high level of assurance that your templates will always remain compatible with AWS services, even if Amazon makes changes to its services.

**Conclusions:**

* Though we find CDK more beneficial than Cloud formation, due to the project timeline and limited knowledge about CDK, Cloud formation remained our choice for code as infrastructure.
* As we chose to work mainly with AWS resources, Cloud formation stands the best option.
* Our project needs to be worked with DevOps and GitOps and moreover DevOps and GitOps best practices, using widely adopted processes such as starting with a git repository and deploying through a CI/CD pipeline can be applied with Cloud formation.

**APPLICATION ENVIRONMENT**

**They are currently hosting a customer facing web application on their on premise environment based on a NodeJS application behind an NGINX reverse proxy.**

**Elastic Beanstalk :**

* Elastic Beanstalk application is the option we chose as a replacement in this case.
* AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with NodeJs and on familiar servers such as Nginx.
* Elastic Beanstalk automatically handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring.
* Within minutes, your application will be ready to use without any infrastructure or resource configuration work on your part. It provisions and operates the infrastructure and manages the application stack for you. This gives developers more time to concentrate on their application.
* It will also keep the underlying platform running your application up-to-date with the latest patches and updates.
* Elastic Beanstalk automatically scales your application up and down based on your application's specific need using easily adjustable Auto Scaling settings. You have the freedom to select the AWS resources, such as Amazon EC2 instance type, that are optimal for your application.
* *(*[*https://aws.amazon.com/elasticbeanstalk/*](https://aws.amazon.com/elasticbeanstalk/)*)*

**Design Decision**:

We decided to use AWS Elastic Beanstalk to host the NodeJs web application. Elastic Beanstalk is a managed service by AWS. The company only has to provide their code and ElasticBeanstalk will do the rest. AWS Elastic Beanstalk is a managed service by AWS. You can simply upload your code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring. At the same time, you retain full control over the AWS resources powering your application and can access the underlying resources at any time. You only pay for the AWS resources used during application runtime. *(*[*https://aws.amazon.com/elasticbeanstalk/*](https://aws.amazon.com/elasticbeanstalk/)*)*

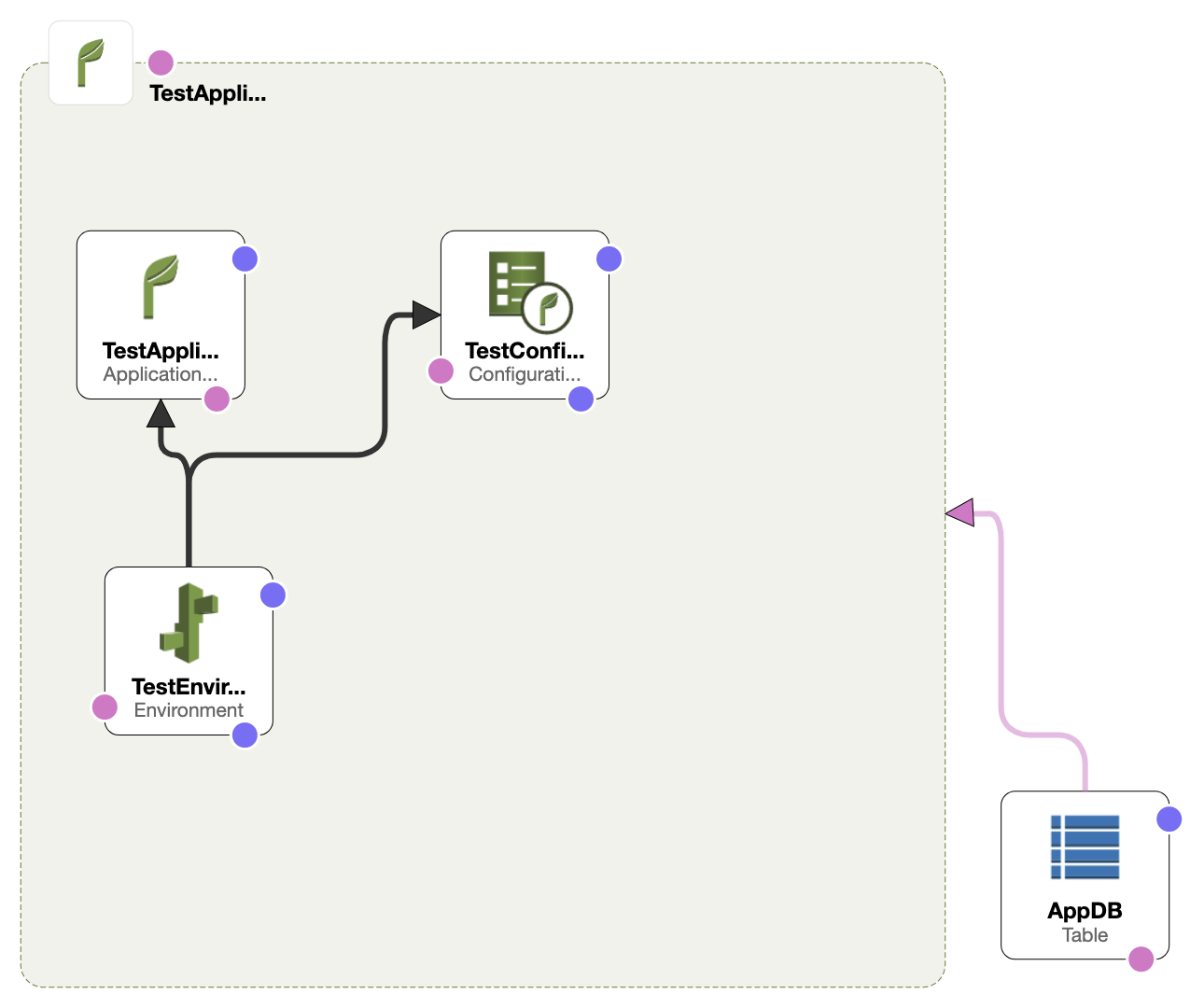
For the application data storage

**Elastic Load Balancer:**

* A fully managed service that enforces highly available configuration by requiring a target subnet in two availability zones, and does not require system administration knowledge.
* On the other hand, NGINX high availability is not an “out of the box feature” in the free version

**Autoscaling :**

AWS Elastic Beanstalk environment includes an Auto Scaling group that manages the Amazon EC2 instances in your environment. In a single-instance environment, the Auto Scaling group ensures that there is always one instance running.



**Company is utilizing a MongoDB cluster for storing data as well as an FTP server for document storage.**

* DocumentDB was also a choice but we decided to go for DynamoDB.
* Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale.
* It's a fully managed, multi-region, multi-active, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications.

**Design Decision:**

To keep costs low and use AWS managed services we decided to use DynamoDB instead of MongoDB cluster for storing the application data and document storage. The assignment clearly stated that the company had enough time to adjust their application before fully deploying into AWS cloud. WIth the use of DynamoDB the application needs to be adjusted.

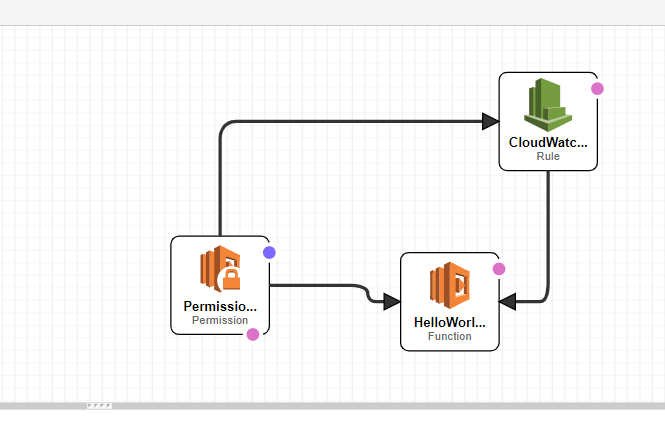
**CRON JOBs**

**Company maintains a cron server, mostly Bash and Python scripts, relevant to a small amount of jobs that need to be executed a few times per day (no more than once per hour).**

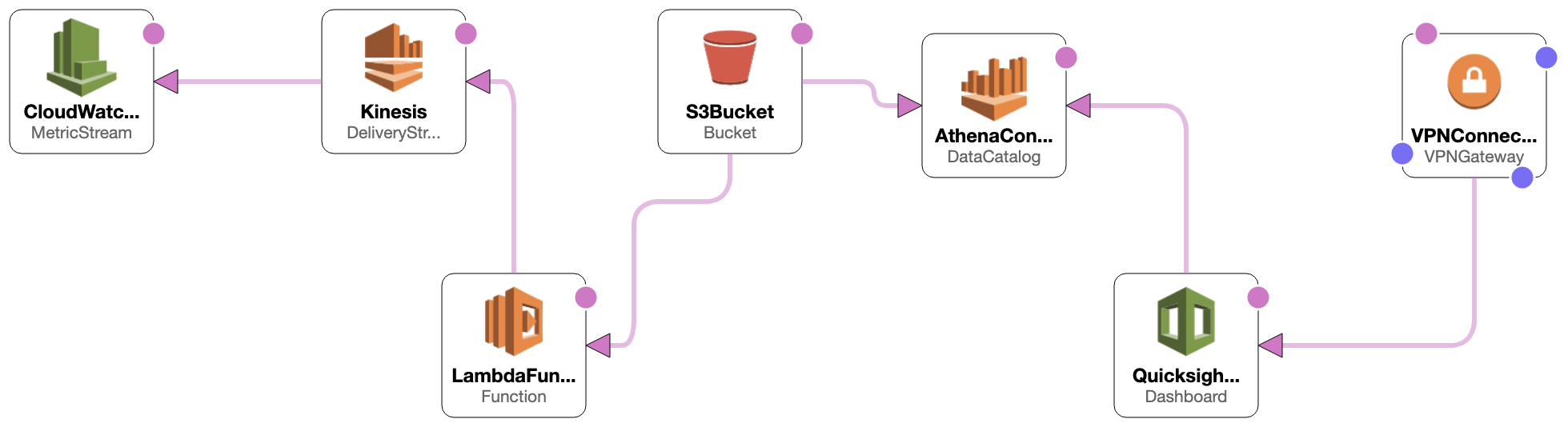
* For this requirement we decided to use Cloudwatch and Lambda functions.
* AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes.
* With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. (*<https://aws.amazon.com/lambda/>).*
* CloudWatch provides you with data and actionable insights to monitor your applications, respond to system-wide performance changes, optimize resource utilization, and get a unified view of operational health.
* You can use CloudWatch to detect anomalous behavior in your environments, set alarms, visualize logs and metrics side by side, take automated actions, troubleshoot issues, and discover insights to keep your application running smoothly.
* You can create rules that self-trigger on an automated schedule in CloudWatch Events using cron or rate expressions. All scheduled events use UTC time zones and the minimum precision for schedules is 1 minute.

**Design Decision:**

We created a CloudWatch Event rule which will trigger a lambda function a few times per day to run the Bash and Python scripts for the small amounts of jobs. This would save costs and is in line with the utilization of managed services as much as possible. The alternatief is running a server on an AWS EC2 instance but that would be costly.



**Design Overview:**



**There is a hard requirement for exporting all application and infrastructure logs to an ElasticSearch Cluster. The customer needs to have access to the Kibana dashboard within their headquarters but the cluster/dashboard *should not* be publicly accessible.**

**CloudWatch metric streams:**

* CloudWatch Metric Streams are fully managed and very easy to set up.
* This is highly scalable and far more efficient.
* In order to make it easier to gain access to CloudWatch metrics faster and at scale, we used CloudWatch Metric Streams.
* Instead of polling (which can result in 5 to 10 minutes of latency), metrics are delivered to a Kinesis Data Firehose stream.
* You can stream metrics to a Kinesis Data Firehose
* that writes data to an endpoint owned by an AWS Partner.
* Or you can stream metrics to a Kinesis Data Firehose of your own.
* From there you can apply any desired data transformations, and then push the metrics into Amazon Simple Storage Service (Amazon S3) or Amazon Redshift.

**Kinesis Firehose:**

* Amazon Kinesis Data Firehose an easy way to reliably load streaming data into data lakes, data stores, and analytics services.
* It can capture, transform, and deliver streaming data to Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, generic HTTP endpoints
* And service providers like Datadog, New Relic, MongoDB, and Splunk.
* It is a fully managed service that automatically scales to match the throughput of your data and requires no ongoing administration.

**Lambda Function:**

* The cron jobs only need to be executed a few times a day, it is cheaper than leaving it on all day.
* Lambda function used to request additional, customized processing of the data before it is sent downstream, which leads to a more self managed system for the client

**S3 Bucket**

* If you want to store logs for long-term; S3 is the right solution.. You can still do processing on S3 using Athena.
* When you import data into a dataset rather than using a direct SQL query, it becomes *SPICE data* because of how it's stored. *SPICE* is the Amazon QuickSight *Super-fast, Parallel, In-memory Calculation Engine*. It's engineered to rapidly perform advanced calculations and serve data.
* Amazon Athena allows you to analyze data in S3 using standard SQL, without the need to manage any infrastructure.

**Athena**

* Amazon Athena is an interactive query service that makes it easy to analyze data directly from Amazon S3 using standard SQL, without the need to manage any infrastructure.
* Athena is serverless, so there is no infrastructure to set up or manage and you can start analyzing your data immediately.

**Quicksight**

* QuickSight lets you easily create and publish interactive business intelligence dashboards in a fully managed way that include Machine Learning-powered insights.
* QuickSight dashboards can be accessed from any device, and seamlessly embedded into your applications, portals, and websites.
* QuickSight is serverless and can automatically scale to tens of thousands of users without any infrastructure to manage or capacity to plan for you.
* Only pay when your users access their dashboards or reports,making it cost-effective for large scale deployments.

**Design Decision:**

The following architecture for the solution consists of a subscription filter for CloudWatch Logs to continuously send the application logs to a Kinesis Firehose data stream. With a Lambda the data will be transformed and populated into a S3 Bucket. Amazon Athena can be used to connect the S3 bucket and Amazon Quicksight. Amazon Quicksight will be in a private subnet and will not be accessible to the public. Quicksight dashboard will be privately accessible through a VPN Gateway connected to the VPC/private subnet

### **VPC per environment setup**

#### **PROS:**

1. **You have a clear separation between your environments due to separate VPC’s.**
2. **You will have finer access control on your environment as the access rules for VPC will effectively be access rules for your environments.**
3. **As an admin it gives you a clear picture of your environments and you have an option to clone you complete environment very easily.**

#### **CONS:**

1. **As mentioned in pros of Single VPC setup you are at some financial loss as you would be duplicating admin application’s across environments**

**In my opinion the decision of choosing a specific set-up largely depends on the scale of your environment if you have a small or even medium sized environment then you can have your infrastructure set-up as “All environments in single VPC”, in case of large set-up I strongly believe that VPC per environment set-up is the way to go.**

Let me know your thoughts and also the points in favour or against of both of these approaches.