Hand-in assignment 3 – Cloud orchestration

Maciej Owczarek 579390, Marcin Grawiński 579370

# Meeting the Learning Objectives

# **3.1. Provision a set of resources using Terraform in a multi-cloud environment**

Most resources are created using Terraform both in AWS and Google Cloud. The only exception is the reverse proxy server as it required a set of outputs from GKE that terraform doesn’t provision.

# **3.2. Deploy a microservices solution on Kubernetes**

The solution on Google Cloud is orchestrated by Kubernetes.

# **3.3. Configure a Kubernetes cluster using Ansible**

It is provisioned by Ansible using a playbook and a deployment file.

# Meeting the Requirements

**REQ-15 : "The AWS-deployment of Cloud Formation stacks should be done using Terraform. "**

All AWS resources except the last one are provisioned using terraform in an automated fashion.

**REQ-16 : "The CloudShirt website should be deployed to Google Cloud Platform using Terraform too. (Docker Compose and/or RDS may be used)"**

We use single file structure for both AWS and Google Cloud resources, there is a single RDS for both providers.

**REQ-17 : "Users can access the CloudShirt website at one external IP-address"**

We managed it by creating a reverse proxy with nginx that redirects users from its external ip to either AWS or Google cloud load balancers

**REQ-18 : "Docker images of the CloudShirt website should be hosted on Artifact Registry."**

The buildMaster EC2 instance pushed the same image to both ECR and Artifact Registry

**REQ-19 : "A Kubernetes cluster should be deployed on Google Cloud Platform in an automated manner. (GKE may be used)."**

GKE cluster is deployed using Terraform with Ansible

**REQ-20 : "The Kubernetes cluster consists of a Master that manages a Slave consisting 5 replica's of the CloudShirt website"**

GKE provides us with a master node and we create 3 worker nodes on which there are 5 replicas of our deployment.

**REQ-21 : "Ansible is used to configure the Kubernetes cluster.**

Ansible configures the GKE cluster with a playbook and a deployment file.

**REQ-22 : "Ansible is used to collect logfiles from the CloudShirt website on AWS or Google Cloud Platform"**

Another Ansible playbook collects logs from the Kubernetes cluster on GKE.

## Most Important Choices

First choice to made was how to get the image to Artifact Registry. First option was to use the buildMaster EC2 instance that already builds the image and pushes to ECR. This solution requires that the instance is authorised with modifying AR. Other options were for example to copy the image from ECR using GCloud VM or even clients pc or to build it from scratch in Google Cloud. We opted for the first option as it doesn’t create additional resources and using a service account on google and putting the key file via ssm securestring, we could securely give access to the AR to buildMaster.

The previous iteration used an RDS with local connection in VPC and now we needed a database that both the EC2 and GCloud VMs could use. We could potentially use two desynchronized databases but that would be insecure and adding unnecessary resources. We made the RDS internet facing but we specified in its security group the internal AWS VPS and all GKE nodes as the only instances that can connect to the DB.

The last choice was how to create a single external IP-address. There are a lot of options: IP Anycast, AWS Global Accelerator, Cloudflare for DNS Load Balancing, Route 53. But these solutions are complex / expensive / limited due to lab environment. We chose to use an EC2 instance as a reverse proxy server using nginx. This way is not the cheapest but still nginx provides good functionality and we don’t need to invest in a domain.

# How to Roll Out the Solution

To rollout the solution first download all files or clone GitHub repository (<https://github.com/Grawikos/CloudAutomation.git>). The lab environment is expected, otherwise, a LabRole with LabInstanceProfile must be created.

Vars.yaml - id, cluster, region;

.tfvars

AWS CLI must be installed and set up with default region and credentials. In the folder with all files execute

|  |  |
| --- | --- |
| Windows | MacOS/Linux |
| *create.ps1 [-userid “<account user id>”, -bucketname “<name>”]* | *deploy.sh [-userid “<account user id>”, -bucketname “<name>”]* |

*<account user id>* - optional, default: found by command  
*<name>* – optional, name used to create a bucket. It will be concatenated with userid to decrease chance of name collision, default: “athena-data-bucket”

After all stacks are created, the link to the website will be outputted to the terminal. It may take a minute to initiate servers.

To deploy static website on the S3 Bucket go to folder with bucketscript.ps1 and website.json and execute.

|  |  |
| --- | --- |
| Windows | MacOS/Linux |
| *bucketscript.ps1 [-userid “<account user id>”, -bucketname “<name>”, region “<region>”]* | *bucketscript.sh [-userid “<account user id>”, -bucketname “<name>” , region “<region>”]* |

*<region>* - region of the user, default: us-east-1

The link to the website will be outputted to the terminal.

To connect to the ElasticStack and see logs of the application as admin, execute.

|  |  |
| --- | --- |
| Windows | MacOS/Linux |
| *connect\_admins.ps1 [-port “<local Port Number>”]* | *connect\_admins.sh [-port “<local Port Number>”]* |

*<local Port Number>* - any open port on the host, default: 8080.

The link to the website will be outputted to the terminal.

On Kibana website (default - localhost:8080), a new dashboard needs to be created in the Discover tab with logs tagged as filebeat (e.g., filebeat-\*).

## Recommendations

**DNS Domain**

The reverse proxy is a solution based on our limits. In a production environment a DNS domain should be bought and e.g. Route 53 applied.

**Improve automation, CI/CD pipeline**

For now little parts of the solution have to be manually modifies. All that should be automated to a single commands. Also filling the variables could be improved.

### **Avoiding Single Points of Failure**

The RDS is crucial to the application, so having only one instance without any redundance is a bad practice and should be fixed by making copies of it.

### **Move application session storage to a database**

Right now the application stores a session in memory and a load balancer keeps track of which client should go to which instance. This is bad as if an instance is deleted, the session will be terminated.

### **Secure Access & Data Protection**

Implementing user authentication for accessing the S3 bucket and encrypting data in both the S3 bucket and EFS would improve security measures.

### **Optimizing Performance**

Employing a caching solution, like AWS CloudFront, to provide static content would reduce the load on backend systems and improve user experience.

### **Implementation of Policies & Roles**

Defining reliable IAM policies and role-based access controls is crutial. Although the current lab environment limits us from creating these policies, but their implementation would significantly improve security and operational management.