**Introduction**

This document describes a container platform design created for the Deloitte Platform Engineering team.

**Requirements**

* The container platform needs to host a basic containerized Flask web application that displays a webpage with the text ‘Hello World!’.
* The environment should be designed in accordance with best practices for an Internet facing platform.

**Assumptions**

* AWS is an acceptable and suitable platform for this design.
* AWS Fargate meets all anticipated use-cases
* 512 CPU units and 1GB of memory per Fargate task is sufficient for the anticipated application load.
* Container platform needs to be highly available

**Design decisions**

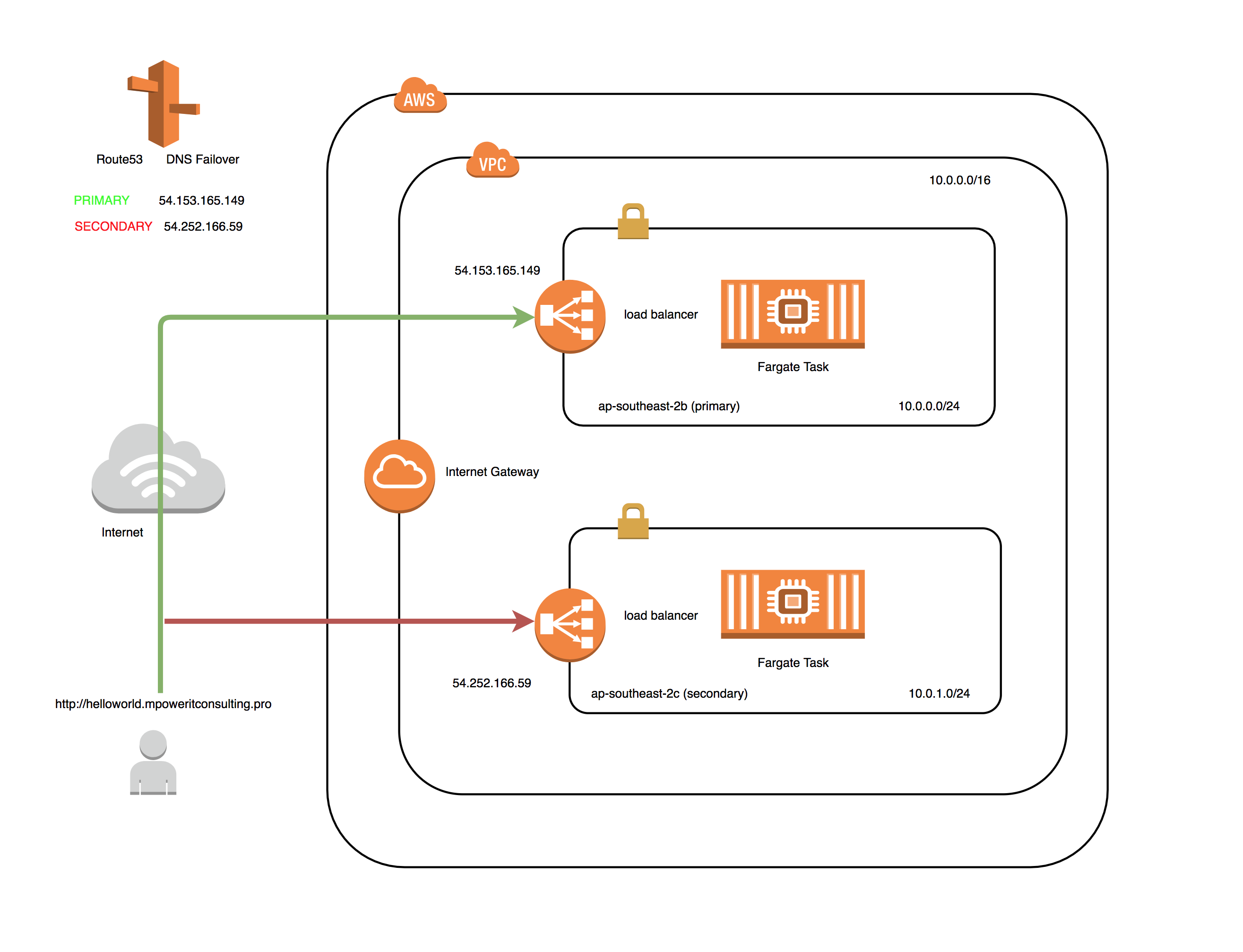
* AWS Fargate is chosen as it does not require the provisioning and management of a container orchestration systems such as Kubernetes, which incurs the least amount of overhead and enables fast deployment.
* For resiliency, multiple containers instances are deployed across two availability zones in the AWS AP-SOUTHEAST-2 region.
* Every availability zone has an Internet-facing network load balancer to distribute incoming load to the containers.
* To protect against failure of an availability zone (data center), AWS Route53 DNS-based failover routing is used to route incoming traffic to the primary site, while failing over to a secondary site if the primary fails a health check.
* Elastic IP (static) are assigned to the load balancers to enable DNS based failover.
* The network load balancers are configured to map incoming traffic on TCP port 80 to TCP port 5000 on the containers, for an improved user experience.
* As per best practice, security groups assigned to the subnets are configured to ensure that only authorized traffic is allowed through to the load balancers, which in this case is TCP on port 80.

**Solution overview**

The highly resilient and scalable platform design is made up of an AWS virtual private cloud (VPC), spanning two availability zones in the AP-SOUTHEAST-2 region. An AWS Fargate cluster is deployed to host the container workloads, which are evenly distributed across the two availability zones.

Application performance and service capacity can be scaled to arbitrary levels by increasing the number of Fargate tasks, to allow more sessions to be served as the need arises.

The solution protects against failure at the container level through the use of health checks, as well as offering site redundancy through the use of Route53 DNS failover routing.



**Traffic Flow**

An end-user opens a web browser and enters the following URL:

<http://helloworld.mpoweritconsulting.pro>

If the health check for primary site is OK, then AWS route 53 will resolve that hostname to the IP address of the load balancer for the primary site, 54.153.165.149 in the ap-southeast-2b availability zone.

Traffic will hit the load balancer configured listener TCP port of 80, after which the load balancer will direct the incoming request to one of the live containers configured in its target group.

The load balancers are configured with health checks that monitors the web service running on the containers for liveliness and checks that the Flask application is working correctly.

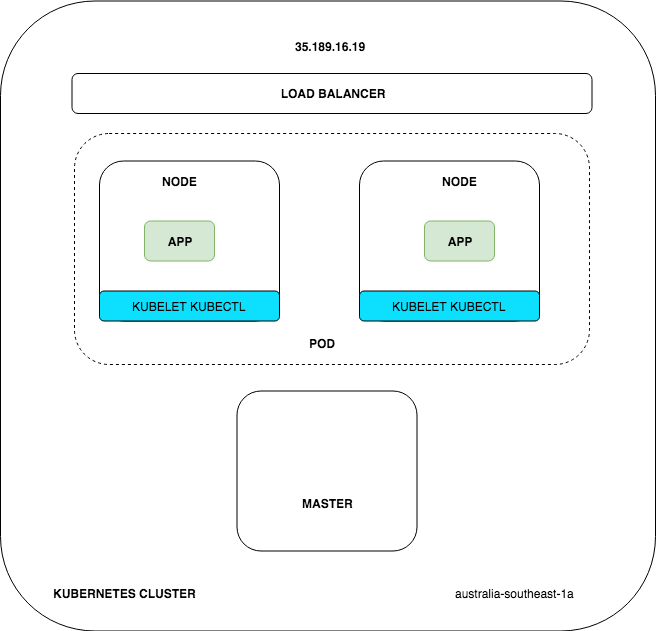
One of the containers that passed the health check will receive the incoming request on TCP port 5000, which will then connect to the Flask application that renders the ‘hello world’ web page. This content is then sent back to the client via the Internet gateway.

If the health check for the primary site fails, then Route53 will re-route traffic to the secondary site, ap-southeast-2c, by changing the IP address that the hostname helloworld.mpoweritconsulting.pro resolves to.

Traffic flow between the end-user and the containers hosted in the secondary site is identical to the first example.

**Kubernetes**

The application has also been deployed on Kubernetes through Google Cloud’s Kubernetes Engine as a 3-node, 2 tasks deployment. Each node has been provisioned with 1 virtual CPU and 3.5 GB of memory each.



The YAML files for this deployment can be found here:

<https://github.com/mlammerse/flask-helloworld/blob/master/helloworld-workloads.yaml>

<https://github.com/mlammerse/flask-helloworld/blob/master/helloworld-service.yaml>

The HelloWorld service is exposed here:

<http://35.189.16.19/>