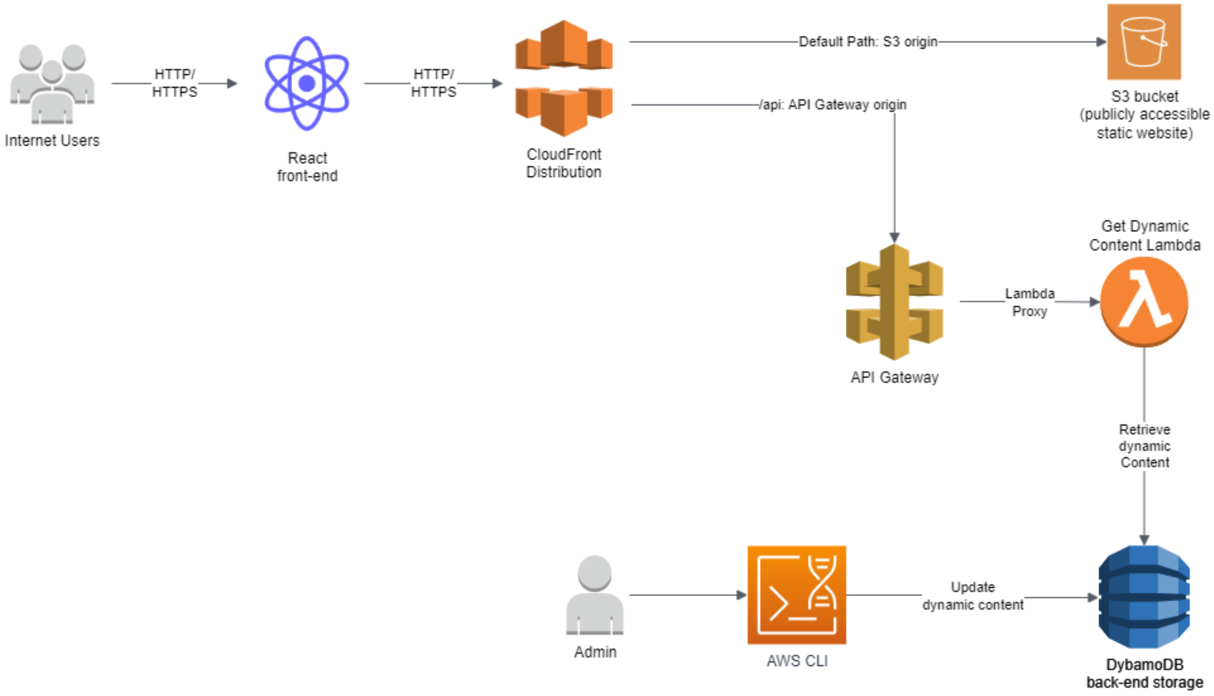
Merapar IAC Interview Technical Challenge

Steven Hunter - 5th December 2022

There are many ways in which a dynamic web page could be implemented using a cloud based solution. Some possibilities would be as follows, along with some pros and cons of each.

High level architecture	**Pros**	**Cons**
Virtual machine in cloud, e.g. EC2 instance	Flexible, can deploy anything in any configuration in one place	Scaling would require multiple VMs, and load balancing e.g. AWS auto-scaling group and ALB Servers need to be patched/upgraded More difficult to secure Need to manage web server configuration Less resilient
Containerised App	Small footprint Can be deployed on Kubernetes/AWS EKS, docker running on VM, or serverless container app e.g. Fargate/ECS More cost effective than VMs Multi-platform	More expensive than other options such as Serverless More complex, especially if managing Kubernetes/EKS
Serverless e.g. AWS Lambda/API Gateway	Minimal deployment Pay as you go Serverless so NO additional infrastructure to manage Scales automatically Resilient	Less portable than containers

Given the relative simplicity of the application I opted for a serverless architecture in AWS as shown…



To summarise, my implementation includes:

- React app deployed to S3 bucket using static website config
- Cloudfront distribution default path requests to the S3 origin, and /api requests to an API Gateway origin
- API Gateway deployment with Demo stage and Lambda proxy integration
- Lambda function with API Gateway trigger which retrieves dynamic page content from a DynamoDB table using DynamoDB.DocumentClient
- Shell script which uses AWS CLI to update dynamic page content in DynamoDB which is then reflected when page is refreshed in React front-end.

Source code including Terraform IaC is at https://github.com/stevenhunter/merapar-iac-demo

Due to limited time, this is not a production-ready implementation. With more time available, I would consider the following additional changes to enhance the solution.

1) Add detailed logging/monitoring e.g. Cloudwatch logs/dashboards/alarms + sns notifications. These would be useful for monitoring response times, load, lambda invocations, DynamoDB throughput etc., and for identifying and responding to issues.

2) Isolate from other workloads by deploying in separate AWS account
3) Restrict to HTTPS access only, and only from CloudFront
4) Prevent external access to the back-end API - only allow requests from CloudFront distribution - require CloudFront custom headers
5) Add CloudTrail logging to detect changes to deployment/config
6) Enable request tracing e.g. XRay
7) Link to custom domain forwarding to CloudFront using Route 53 and ACM to enable TLS encryption through trusted certificates
8) Add a front-end feature enabling updates of dynamic content in DynamoDB. This would require user authentication, potentially using SSO/LDAP/Cognito etc and possibly a POST endpoint which could be invoked from the React front-end by logged in users.