



App backend | Take-home exercise

Backend Engineer, App

October 2023

The goal of this challenge is to assess your applied coding skills as well as your ability to research and apply new concepts. You should expect to spend **3-6** hours on this. This exercise will help you get a better understanding of engineering at TFH. It will particularly test architecture and infrastructure knowledge.

The exercise

The goal is to build a simple app to store a binary Merkle tree and an API to retrieve certain information about the tree. The app should be deployed to AWS using infrastructure-as-code (IaC).



Merkle Trees are a very useful data storage tool which makes data verification and integrity checking efficient. They are widely used in the blockchain space. A binary Merkle tree is a specific case of a Merkle tree in which nodes have to 2 children.

Initial context

Binary Trees

To simplify things, this exercise is only intended to work with binary trees. Given a binary tree:

```
flowchart TD
    subgraph root
        0(["0"])
    end
    subgraph intermediate_nodes
        1(["1"])
        2(["2"])
        3(["3"])
        4(["4"])
        5(["5"])
        6(["6"])
    end
    subgraph leaves
        7(["7"])
        8(["8"])
        9(["9"])
        10(["10"])
        11(["11"])
        12(["12"])
        13(["13"])
        14(["14"])
    end
    0 --> 1
    0 --> 2
    1 --> 3
    1 --> 4
    2 --> 5
    2 --> 6
    3 --> 7
    3 --> 8
    4 --> 9
    4 --> 10
    5 --> 11
    5 --> 12
    6 --> 13
    6 --> 14
```

The single top node is called the *root*, the lowest nodes are called *leaves* and the remaining nodes are called *intermediate nodes*. Layers are counted zero based from

the root, called the *depth*. The number of layers is called the depth of the tree. The nodes are numbered zero-based left-to-right and top-to-bottom called their *index*.

In addition to the index, we can also identify the nodes by their depth and offset from the left:

```
flowchart TD
    0(["(0,0)"])
    1(["(1,0)"])
    2(["(1,1)"])
    3(["(2,0)"])
    4(["(2,1)"])
    5(["(2,2)"])
    6(["(2,3)"])
    0 --> 1
    0 --> 2
    1 --> 3
    1 --> 4
    2 --> 5
    2 --> 6
```

CDK

AWS's CDK is a framework to define cloud infrastructure as code (IaC). You will use CDK to deploy your app.

Task details

1. Set up a project that uses CDK for deployment. Getting started on CDK.



You will need to run `cdk bootstrap aws://ACCOUNT-NUMBER/REGION` once for your AWS account to bootstrap the CloudFormation stack that will deploy your app stacks.

2. Create a data store that lets you persistently store a binary Merkle tree. Your data structure should allow for easy retrieval of specific nodes.

a. Data can be stored in DynamoDB, Redis, S3, RDS, or any other mechanism you see appropriate in AWS. **Be sure to document why you chose a particular storage mechanism.**

b. Your code should initialize the data store. Example:

```
import * as cdk from 'aws-cdk-lib';

const bucket = new cdk.aws_s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket',
  versioned: true,
});
```

c. Loading the entire tree to memory to perform later operations is acceptable for this challenge.

3. Create a simple method that will create a binary Merkle tree and store it in your storage engine. The tree can be static (i.e. you can initialize it once and keep it static) or dynamic.

a. To construct the tree, the hash of its two children must be computed. The tree can be initialized with any initial arbitrary value for the leaves.

$$\text{hash}(\text{node}) = \text{SHA3}(\text{hash}(\text{leftChild}(\text{node})) || \text{hash}(\text{rightChild}(\text{node})))$$

4. Write a function that given an `index` of a node, returns its `depth`, `offset` and value (i.e. either the leaf value or the hash of the node).




5. Create an API route that exposes your function from the previous step as a simple API route. One way to do this is with a lambda function deployed through CDK ([Hello Lambda CDK workshop](#)).

```
// Non-normative request & response below

POST <endpoint>/retrieve
Content-Type: application/json
{
  "index": 5
}
```

```
Response:
{
  "depth": 2,
  "index": 2,
  "value": "4d741b6f1eb29cb2a9b9911c82f56fa8d73b04959d3d9d222895df6c0b28aa15"
}
```

Evaluation criteria

-  **Prioritization.** Did you prioritize the most important things? Was your time spent efficiently? Does the project accomplish the tasks?
-  **Technical depth.** Is the solution efficient and scalable? Does it follow best practices? Is your technical design fit for the task requirements?
-  **Clean code & documentation.** Is the code simple and easy to understand? Is it structured to be maintainable over time? Is everything well documented so that anyone can understand the whole context from it? Did you document points that require further investigation and/or optimization?

Submission requirements

- Typescript is the preferred language for this assessment but you can also deliver it in Python, Rust or Javascript if it better helps accomplish your goals.
- For deployment, CDK is strongly preferred but you can use alternative IaC options like Terraform, CloudFormation, Chalice, SST, CF Wrangler.
- Testing is recommended but optional; prioritize where your time is best spent.
- You should document your project in the README, it should be clear what it is and how it works. Clear instructions on how to run the project must be included.
- Please take note on any friction points or complexities you experience along the way, we'll discuss them in a follow up interview.

- The submission must be the code repository compressed and sent using the link provided in the email from your recruiter.