Terraform Provider Schema Issue

Problem: When certain complex data configurations are included in the schema during compilation, an issue occurs with the provider module that fails due to OOM. When additional memory (e.g. swap) is provisioned, evidence suggests that the substantial size of the compiled binary is attributed to the direct inclusion of a deeply nested object hierarchy, leading to an the inclusion of extremely large stack trace debug data (stored in the .rodata section of the ELF binary). We suspect the data volume to be non-linear and a significant factor contributing to the problem.

To address this performance concern, one potential solution involves flattening the nested map structure. By creating the most deeply nested maps first and storing them in an array, future maps can reference this array, effectively eliminating the nested code structure while preserving the original map's hierarchy. This solution aims to optimize object generation during compile time and reduce the compiled binary size.

```
dataMap := map[string]interface{}{
    "a": map[string]interface{}{
        "b": map[string]interface{}{
            "c": "hello world"
        }
    }
}

// would get flattened to...

var flattened = make([]interface{}, 3112)
flattened[0] := schema.Attribute{"c":"hello world"}
flattened[1] := schema.Attribute{"b":flattened[0]}
flattened[2] := schema.Attribute{"a":flattened[1]}
return flattened[2]
```

Alternatively, another solution proposes generating objects at runtime instead of compile time. In this approach, the generator code would use a JSON file to dynamically generate the attribute hierarchy as objects in memory. By doing so, the size of the binary would be significantly reduced. While this method may involve a slight trade-off, as objects which are generated during runtime can cannot be pre-compiled, the impact of iterating through the map depth during runtime is expected to be minimal. The JSON file could be directly embedded and stored into the compiled binary for easy distribution. Unit or integration tests can be executed in order to confirm code-over-schema health.

```
stringAttributes := make([]schema.StringAttribute, 1*1024*1024) // force heap
attributes := make([]schema.Attribute, 1*1024*1024)
attributes[0] = schema.ListNestedAttribute{NestedObject: schema.NestedAttributeObject
   "x": stringAttributes[0],
}}}
// example of generating objects dynamically at runtime
// real code would iterate the JSON and emit commensurate objects
for i := 1; i < 1000; i++ {
   attributes[i] = schema.ListNestedAttribute{NestedObject: schema.NestedAttributeObj
       "nested": attributes[i-1],
   }}}
}
return attributes[len(attributes)-1]
```

	Nested Compile Time (i.e. current code-gen format)	Flattened Compile Time	Runtime
Example source Code Size	1.5MB	1.9MB	753bytes
Binary Size	35.7MB	21.5MB	3.2MB