Better CloudFormation with Jsonnet

Andrea Bedini - KZN Group

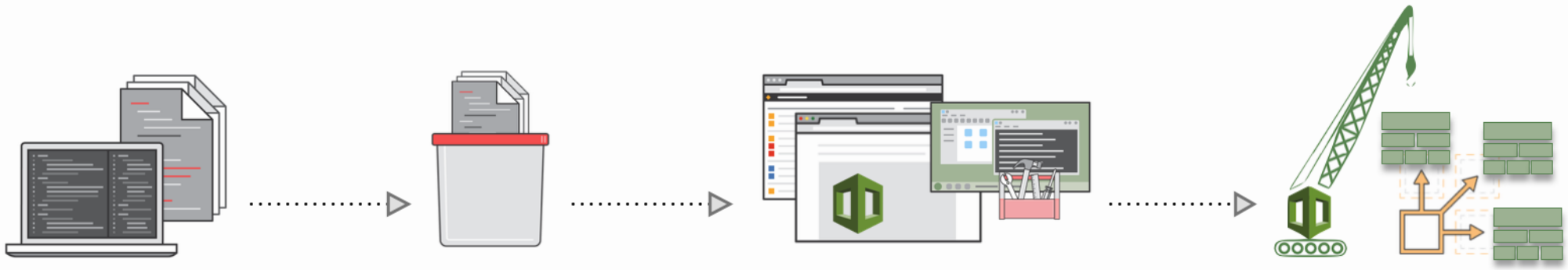
KZN Group

> Our mission is to simplify the lives of our customers.

- We like lego blocks
- Reusable pieces of work
- Avoid repeating / redeveloping
- AWS only

AWS CloudFormation

> AWS CloudFormation provides a common language for you to describe and provision all the infrastructure resources in your cloud environment.

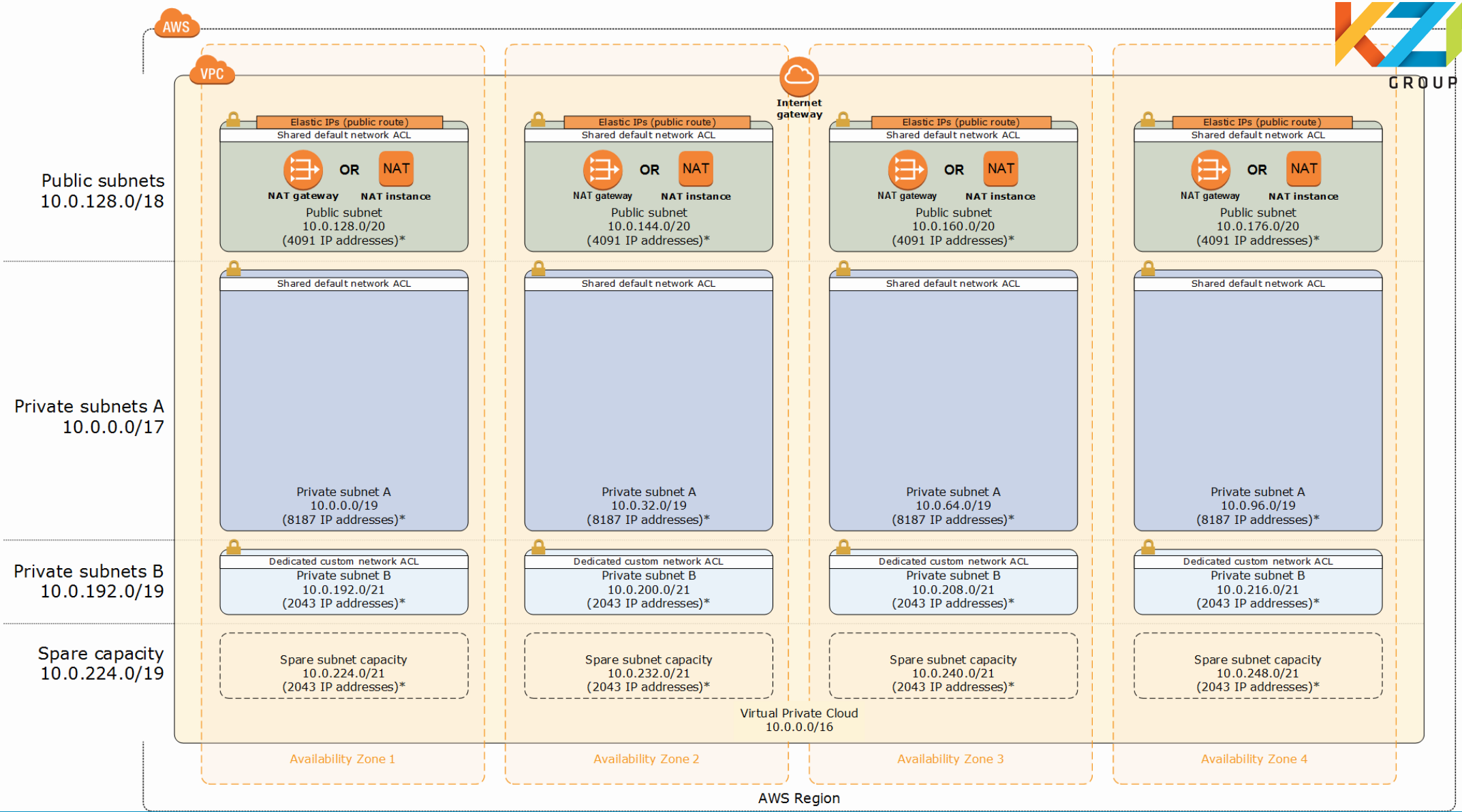


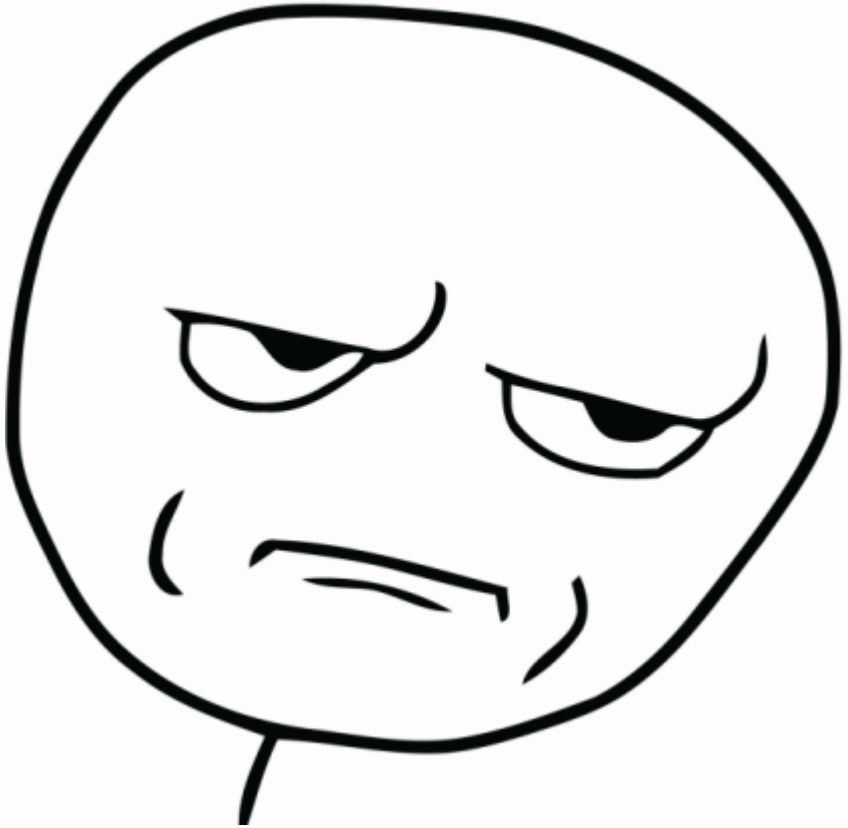
Code your infrastructure from scratch with the CloudFormation template language, in either YAML or JSON format, or start from many available sample templates

Check out your template code locally, or upload it into an S3 bucket

Use AWS CloudFormation via the browser console, command line tools or APIs to create a stack based on your template code

AWS CloudFormation provisions and configures the stacks and resources you specified on your template


AWS
VPC
Internet gateway
Public subnets
10.0.128.0/18
Private subnets A
10.0.0.0/17
Private subnets B
10.0.192.0/19
Spare capacity
10.0.224.0/19
Elastic IPs (public route)
Shared default network ACL
NAT gateway
OR
NAT instance
Public subnet
10.0.128.0/20
(4091 IP addresses)*
Public subnet
10.0.144.0/20
(4091 IP addresses)*
Public subnet
10.0.160.0/20
(4091 IP addresses)*
Public subnet
10.0.176.0/20
(4091 IP addresses)*
Private subnet A
10.0.0.0/19
(8187 IP addresses)*
Private subnet A
10.0.32.0/19
(8187 IP addresses)*
Private subnet A
10.0.64.0/19
(8187 IP addresses)*
Private subnet A
10.0.96.0/19
(8187 IP addresses)*
Dedicated custom network ACL
Private subnet B
10.0.192.0/21
(2043 IP addresses)*
Private subnet B
10.0.200.0/21
(2043 IP addresses)*
Private subnet B
10.0.208.0/21
(2043 IP addresses)*
Private subnet B
10.0.216.0/21
(2043 IP addresses)*
Spare subnet capacity
10.0.224.0/21
(2043 IP addresses)*
Spare subnet capacity
10.0.232.0/21
(2043 IP addresses)*
Spare subnet capacity
10.0.240.0/21
(2043 IP addresses)*
Spare subnet capacity
10.0.248.0/21
(2043 IP addresses)*
Virtual Private Cloud
10.0.0.0/16
Availability Zone 1
Availability Zone 2
Availability Zone 3
Availability Zone 4
AWS Region



Infrastructure as Code?

Now you have all the coding problems:

- DRY (Don't-Repeat-Yourself)
- Reusability
- Readability

DRY (Don't-Repeat-Yourself)

```
"PrivateSubnet1A": {
    "Condition": "PrivateSubnetsCondition",
    "Type": "AWS::EC2::Subnet",
    "Properties": {
        "VpcId": { "Ref": "VPC" },
        "CidrBlock": { "Ref": "PrivateSubnet1ACIDR" },
        "AvailabilityZone": {
            "Fn::Select": [ "0", { "Ref": "AvailabilityZones" } ]
        },
        "Tags": [
            { "Key": "Name", "Value": "Private subnet 1A" },
            { "Key": "Network", "Value": "Private" }
        ]
    }
},
```

Reusability [1/2]

AWS::Include transform

```
Transform:
  Name: AWS::Include
  Parameters:
    Location: s3://...
```

Nested stacks

```
MyComponent:
  Type: AWS::CloudFormation::Stack
  Parameters: ...
```

- Includes have many non-obvious limitations
- Nested stacks are brittle / costly

Reusability [2/2]

Limited support for conditional logic

```
"Conditions": {
    ...
    "3AZCondition": {
    "Fn::Or": [
        { "Fn::Equals": [ { "Ref": "NumberOfAZs" }, "3" ] },
        { "Condition": "4AZCondition" }
    ]
    },
    "4AZCondition": {
    "Fn::Equals": [ { "Ref": "NumberOfAZs" }, "4" ]
    },
},
```

CloudFormation as Language?

KZN GROUP

- Painful to hand craft
- Limited support for code reuse
  - cut & paste is ☹️
- Limited support for conditional logic
- Hard to rationalise about CF templates
  - flat structure, AKA wall-of-text

I KNOW THAT FEEL BRO

Use a generator they say!

- Use a DSL to compose a template in another programming languages (Python, Javascript, Scala, Haskell, ...)

Jsonnet

- Little functional programming language
- Developed by a Google employee as a 10% project
- Simple extension to JSON syntax, familiar to everyone
- Many syntax improvement over JSON

- Databricks, Declarative Infrastructure with the Jsonnet Templating Language

:lightbulb:

```
"PrivateSubnet1A": {
  "Type": "AWS::EC2::Subnet",
  "Properties": {
    "VpcId": { "Ref": "VPC" },
    "CidrBlock": { "Ref": "PrivateSubnet1ACIDR" },
    "AvailabilityZone": {
      "Fn::Select": [ "0", { "Ref": "AvailabilityZones" } ]
    },
    "Tags": [
      { "Key": "Name", "Value": "Private subnet 1A" },
      { "Key": "Network", "Value": "Private" }
    ]
  }
}
```

```
// jsonnet!
local PrivateSubnet(cidr, az, tag) = {
  Type: 'AWS::EC2::Subnet',
  Properties: {
    VpcId: { Ref: 'VPC' },
    CidrBlock: cidr,
    AvailabilityZone: {
      'Fn::Select': [az, { Ref: 'AvailabilityZones' }],
    },
    Tags: [
      { Key: 'Name', Value: 'Private subnet ' + az + tag },
      { Key: 'Network', Value: 'Private' },
    ],
  },
};
```

```
// jsonnet!
local Subnet(cidr, visibility, az, tag='') = {
  Type: 'AWS::EC2::Subnet',
  Properties: {
    VpcId: { Ref: 'VPC' },
    CidrBlock: cidr,
    AvailabilityZone: {
      'Fn::Select': [az, { Ref: 'AvailabilityZones' }],
    },
    Tags: [
      { Key: 'Name', Value: visibility + ' subnet ' + az + tag },
      { Key: 'Network', Value: visibility },
    ],
    [if visibility == 'Public' then 'MapPublicIpOnLaunch']: true,
  },
};
```

```
local subnetName(visibility, az, tag='') =
  visibility + 'Subnet' + az + tag;

{
  Resources:
    {
      // Private subnets
      local name = subnetName('Private', az, tag),
      [name]: Subnet({ Ref: name + 'CIDR' }, 'Private', az, tag)
      for az in std.range(1, 4) for tag in ['A', 'B']
    } + {
      // Public subnets
      local name = subnetName('Public', az),
      [name]: Subnet({ Ref: name + 'CIDR' }, 'Public', az)
      for az in std.range(1, 4)
    },
}
```

Real world example

- KZN Jumpstart
  - AWS Landing Zone-like environment
  - one-click deploy[a]
- Auditing / security / monitoring baseline
- Multiple components
- Written in Jsonnet

[a] conditions apply

KZN Jumpstart - Threat Detection

- Based on AWS Guard Duty
- Multiple accounts / regions
- Jsonnet template
  - reads configuration file
  - renders customised CloudFormation template

```
function(memberAccounts) {
  Resources: {
      // Need to activate master detector
      GuardDuty: GuardDuty.EnabledDetector
    } + {
      // One member for each account in the organisation
      ['GuardDutyMember' + memberAccount.name]:
        guardDutyMemeber(memberAccount, 'GuardDuty')
      for memberAccount in memberAccounts
    } + {
      // One Event Rule for severity level
      ['GuardDuty%sSevFindingEventRule' % sev.severity]:
        eventRule(
          description='Guard Duty + ses.severity,
          pattern=sev.eventPattern
        )
      for sev in findingSeverityMaps
    },
}
```

Pain points

- 👎 Just a language, no "batteries included"
  - We use CloudFormation specs to generate a libray of low-level components (1-to-1 with CF resources types)
- 👎 Template validation (no types)
  - Rely on external tools
- 🖐 Developing reusable components is not trivial

Closing

- Our speed in developing large stacks improved significantly
- Near-zero adoption cost

col-6[

"Introducing AWS CloudFormation Macros"

https://aws.amazon.com/about-aws/whats-new/2018/09/introducing-aws-cloudformation-macros/

AWS CloudFormation Macros in 1 min

- CloudFormation will pass a template through a lambda function
- can pre-process a fragment or whole template
- same technology that supports SAM

```
Transform: 'AWS::Serverless-2016-10-31'
```

```
AWSTemplateFormatVersion: "2010-09-09"
Transform: [Jsonnet]
Resources:
  Table: |
    #!jsonnet
    dynamoDb.Table(readUnits=10, writeUnits=5)
  S3Bucket:
    Type: "AWS::S3::Bucket"
    Properties:
      Tags: |
        #!jsonnet
        to_entries({
          Project: 'take-over-the-world',
          Stage: 'testing'
        })
```

Grab it while it's fresh

https://github.com/KZNGroup/cf-macro-jsonnet