7️⃣

Lecture 7 - IAC - Cloudformation, AWS CDK, Other IaC tools (1h)

- Q&A about the previous lesson (3-5m)

IaC Concepts

- Problems solved
  - inconsistency
  - inability to reproduce the configuration
  - human error when creating the infrastructure manually
- Benefits delivered
  - consistent infrastructure
  - accountability - change introduced in the IaC code is easily detectable
  - documentation - docs and readme-s can be autogenerated based on code and the code itself can serve as a documentation
  - easier to comply with the security standards
  - store, version, test the infrastructure in your source code repo
  - visibility over the environment without launching or accessing it
  - possibility to deploy infrastructure at high speed and large scale
  - can be automated with CI/CD
  - declarative syntax
  - infrastructure is no longer an ancient knowledge if stored in a code
- Caveats ⚠️
  - once IaC - be always IaC to avoid drifts in the environments
  - price of an error can be much higher

Cloudformation

- Pros of using CloudFormation → https://aws.amazon.com/cloudformation/faqs/
  - Native AWS solution - other services often use it for resources deployment

    - yaml and json syntax supported
    - bunch of ready templates available → https://aws.amazon.com/cloudformation/resources/templates/
    - Built-in visualiser and changes preview with ChangeSets
    - automatic dependency management
    - supports pretty much every resource → AWS resources
    - features:
        - automatic rollbacks
        - helper scripts (kinda like hooks) https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-helper-scripts-reference.html
            - cfn-init
            - cfn-signal
            - cfn-get-metadata
            - cfn-hup
        - hooks → https://aws.amazon.com/blogs/mt/proactively-keep-resources-secure-and-compliant-with-aws-cloudformation-hooks/
    - Custom resources are supported → https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/registry.html
    - It's free, but you pay for the resources it created
- Template anatomy → https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-anatomy.html
    - Only `Resources` block is required
    - mappings → https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/mappings-section-structure.html
    - conditions → https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/conditions-section-structure.html
    - intrinsic functions → https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference.html
    - (others)
- Launching a VPC stack with CloudFormation

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/ebff9876-8fdb-453a-8990-24f4085ea7fd/vpc.yaml

- Cons of using Cloudformation
  - no modules or extensions - writing code is simply copy and paste
  - can be painful to write and debug
  - the templates can grow very big
  - in order to use outputs - templates to be launched one after another
  - sometimes it stuck
- Other topics to check out
  - stackSets
  - nested stacks
  - drift detection
- Other tools:
  - Terraform → https://www.terraform.io/
    - simpler syntax
    - cloud-agnostic (single codebase for the multi-cloud)
    - state management (desired vs actual)
    - functions, loops, conditions
    - fancy features (like provisioners)
    - existing resources import, ability to interact with non-managed resources
    - really extendable, modular, has huge community
    - much easier to maintain
  - Pulumi → https://www.pulumi.com/
  - Ansible/Puppet/Chef

CDK

- Define infrastructure by using one of the available programming languages→ JavaScript, TypeScript, Python, Java, C#, and Go
- CDK examples → https://github.com/aws-samples/aws-cdk-examples

```
import os.path

from aws_cdk.aws_s3_assets import Asset

from aws_cdk import (
    aws_ec2 as ec2,
    aws_iam as iam,
    App, Stack
)

from constructs import Construct

dirname = os.path.dirname(__file__)


class EC2InstanceStack(Stack):

    def __init__(self, scope: Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # VPC
        vpc = ec2.Vpc(self, "VPC",
            nat_gateways=0,
            subnet_configuration=[ec2.SubnetConfiguration(name="public",subnet_type=ec2.SubnetType.PUBLIC)]
            )

        # AMI
        amzn_linux = ec2.MachineImage.latest_amazon_linux(
            generation=ec2.AmazonLinuxGeneration.AMAZON_LINUX_2,
            edition=ec2.AmazonLinuxEdition.STANDARD,
            virtualization=ec2.AmazonLinuxVirt.HVM,
            storage=ec2.AmazonLinuxStorage.GENERAL_PURPOSE
            )

        # Instance Role and SSM Managed Policy
        role = iam.Role(self, "InstanceSSM", assumed_by=iam.ServicePrincipal("ec2.amazonaws.com"))

        role.add_managed_policy(iam.ManagedPolicy.from_aws_managed_policy_name("AmazonSSMManagedInstanceCore"))

        # Instance
        instance = ec2.Instance(self, "Instance",
            instance_type=ec2.InstanceType("t3.nano"),
            machine_image=amzn_linux,
            vpc = vpc,
            role = role
            )

        # Script in S3 as Asset
        asset = Asset(self, "Asset", path=os.path.join(dirname, "configure.sh"))
        local_path = instance.user_data.add_s3_download_command(
            bucket=asset.bucket,
            bucket_key=asset.s3_object_key
        )

        # Userdata executes script from S3
        instance.user_data.add_execute_file_command(
            file_path=local_path
            )
        asset.grant_read(instance.role)

app = App()
EC2InstanceStack(app, "ec2-instance")
```

```
app.synth()
```

- pros and cons
  - great for serverless apps or containers
- this is still compiled into the Cloudformation template → Synth
  https://docs.aws.amazon.com/cdk/v2/guide/cli.html
- do not be confused with SDK
  - CDK - define the infrastructure in code
  - SDK - interact with the infrastructure in code
- Cool CDK hands-on is available at Stefan Maarek's course

Q&A session

- topic discussion
- sharing useful external resources and links