AWS re:Invent

WIN303

# Deploy modern apps with the AWS Cloud Development Kit for .NET Core

**Rahul Chugh**
Sr. Partner Solutions Architect
Amazon Web Services

**Vlad Hrybok**
Sr. Partner Solutions Architect
Amazon Web Services

AWS re:Invent



aws

# Agenda

Application lifecycle on AWS

Infrastructure as Code (IaC)

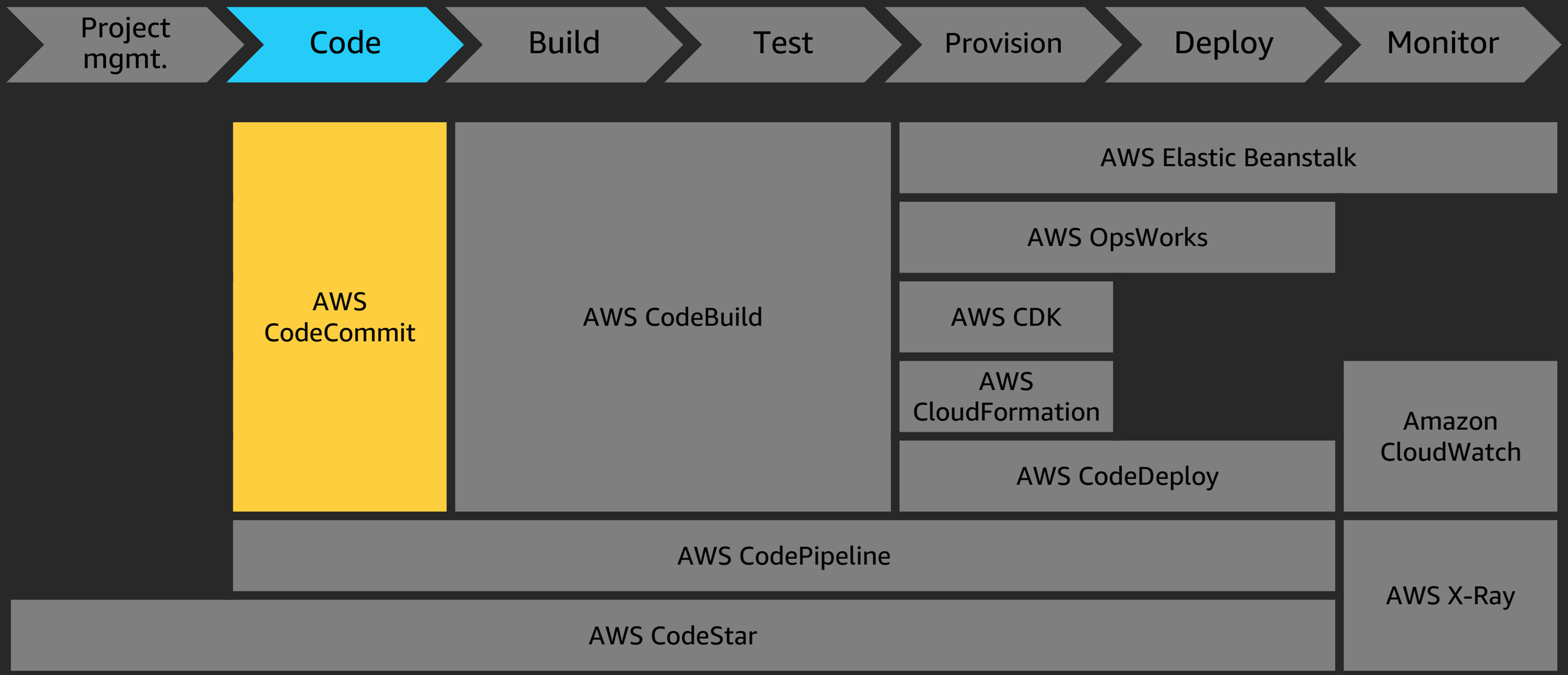
AWS Cloud Development Kit (AWS CDK)

AWS CDK workshop

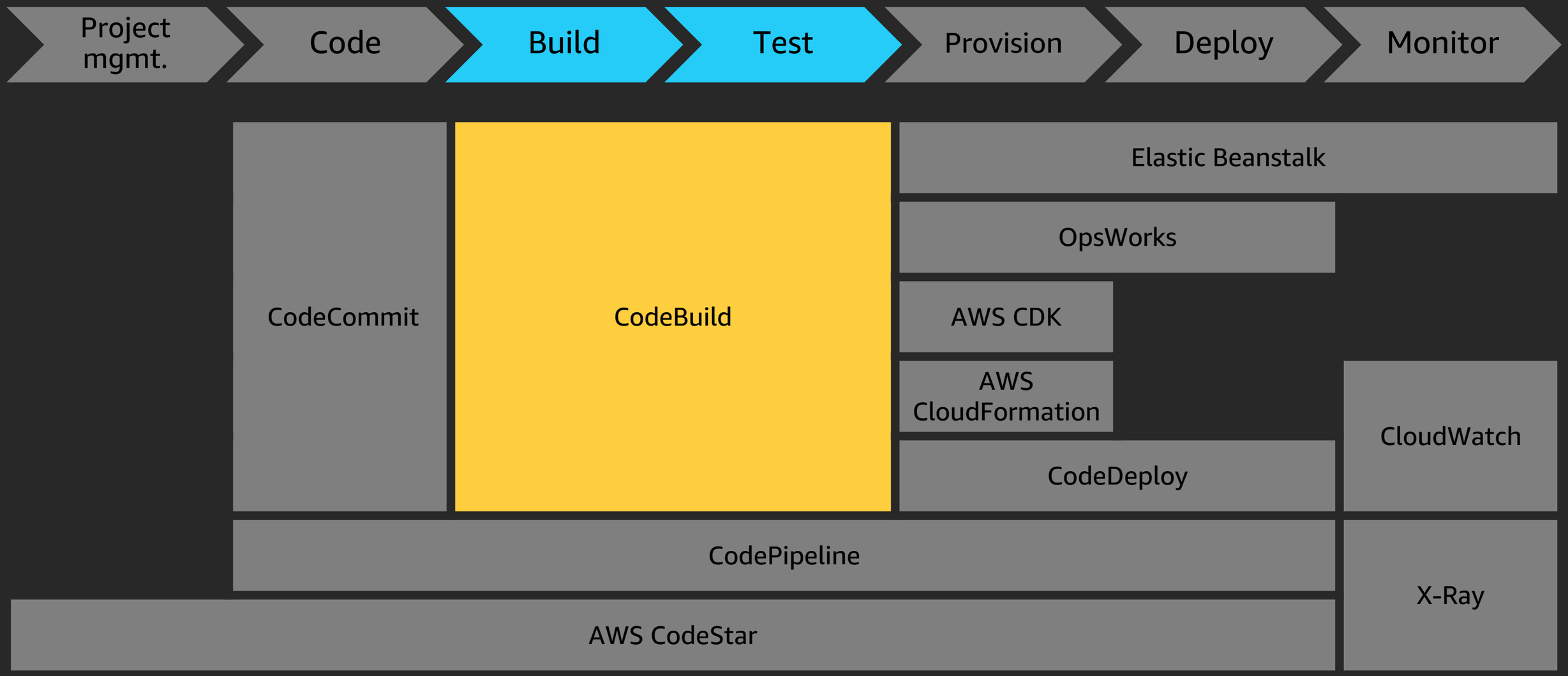
# Application lifecycle on AWS

# Application lifecycle

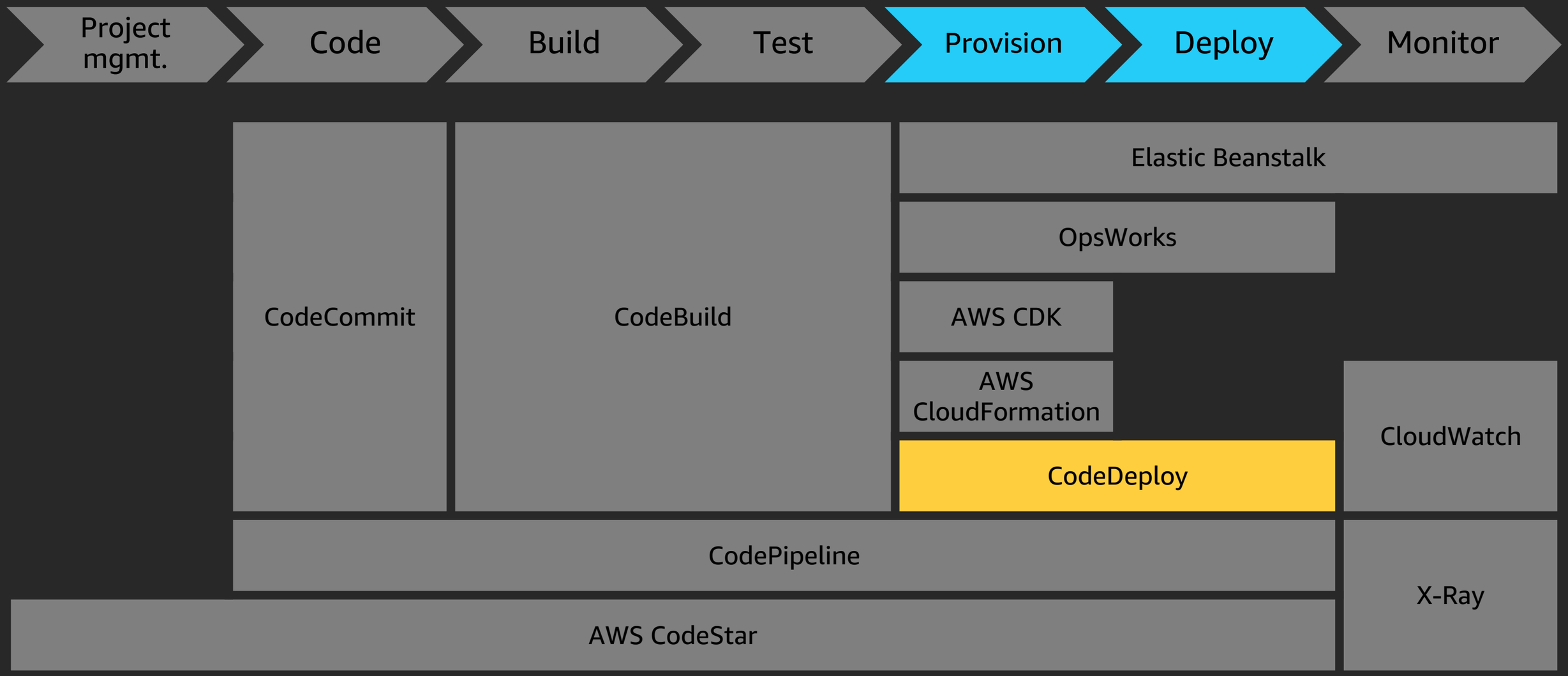
# Application lifecycle on AWS



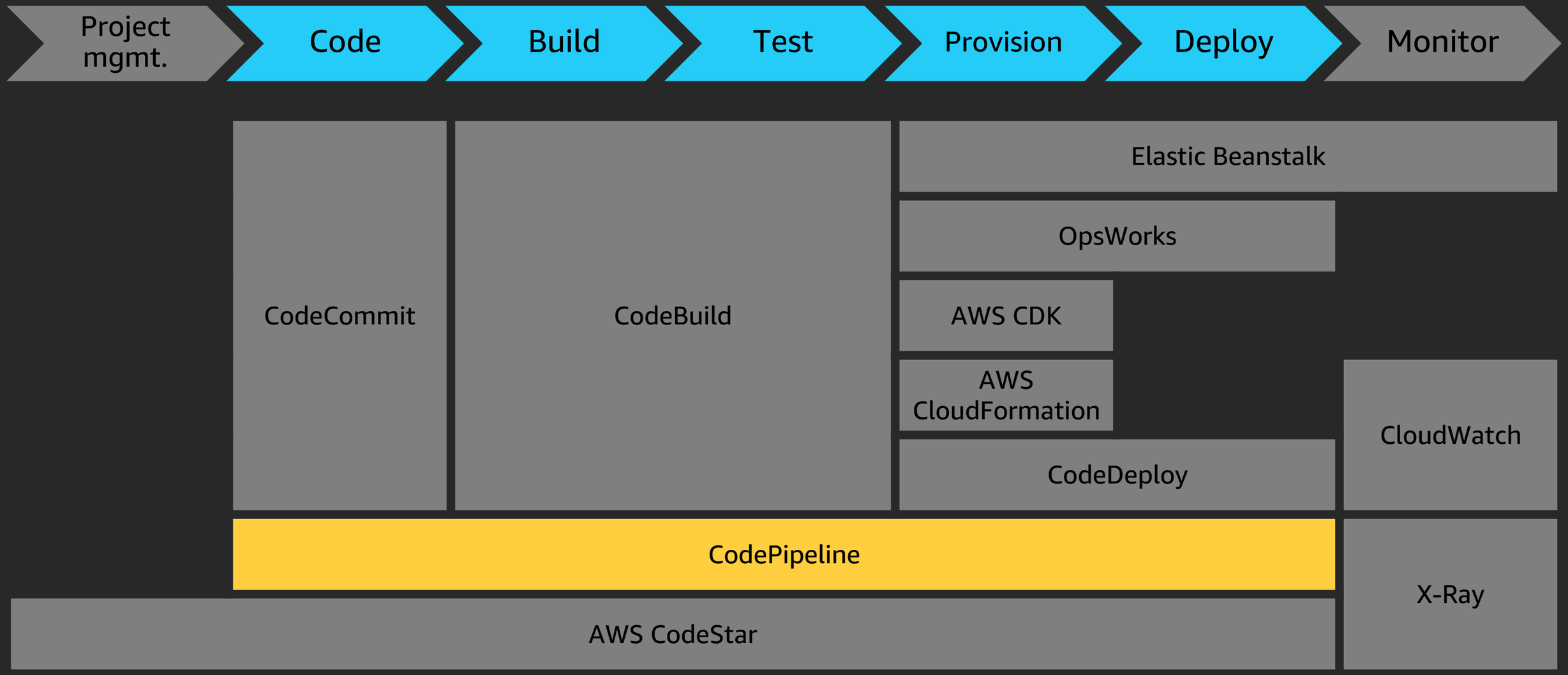
# Application lifecycle on AWS



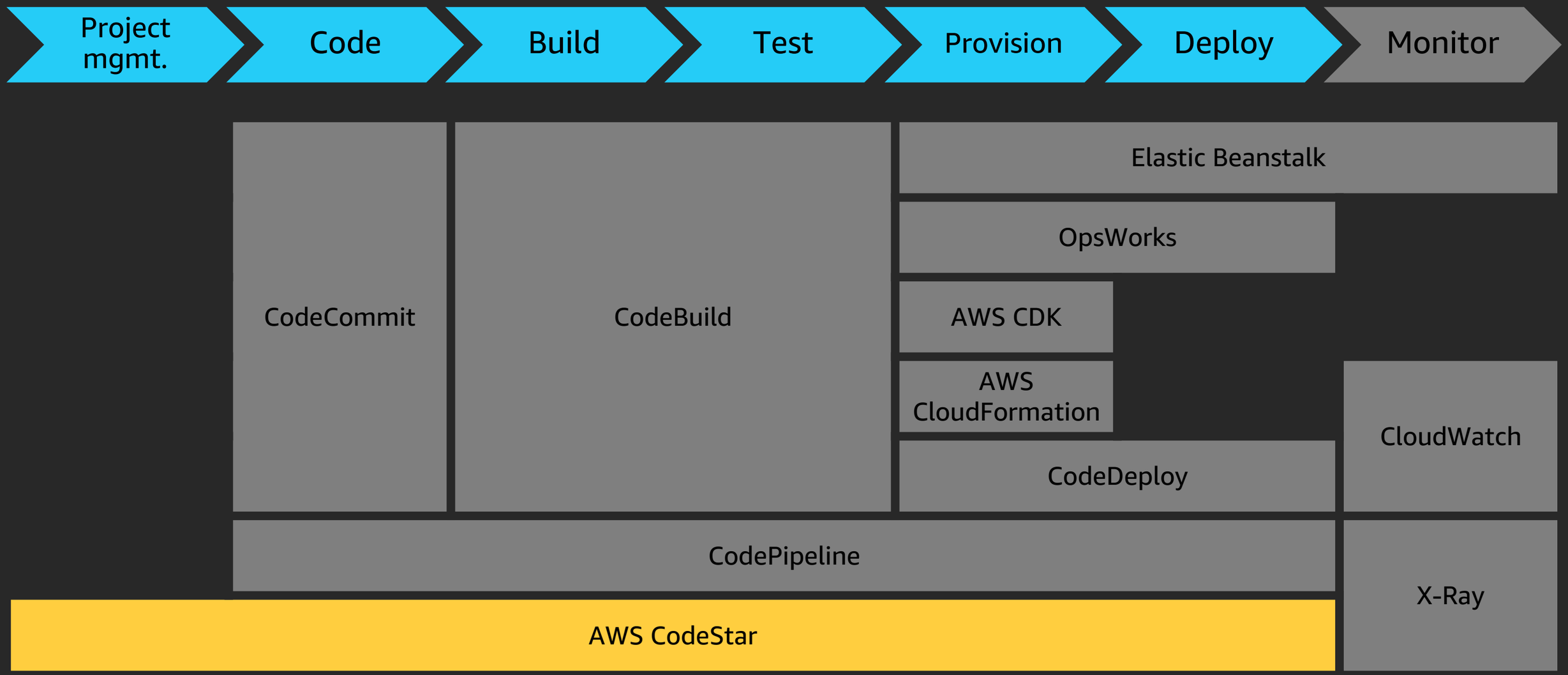
# Application lifecycle on AWS



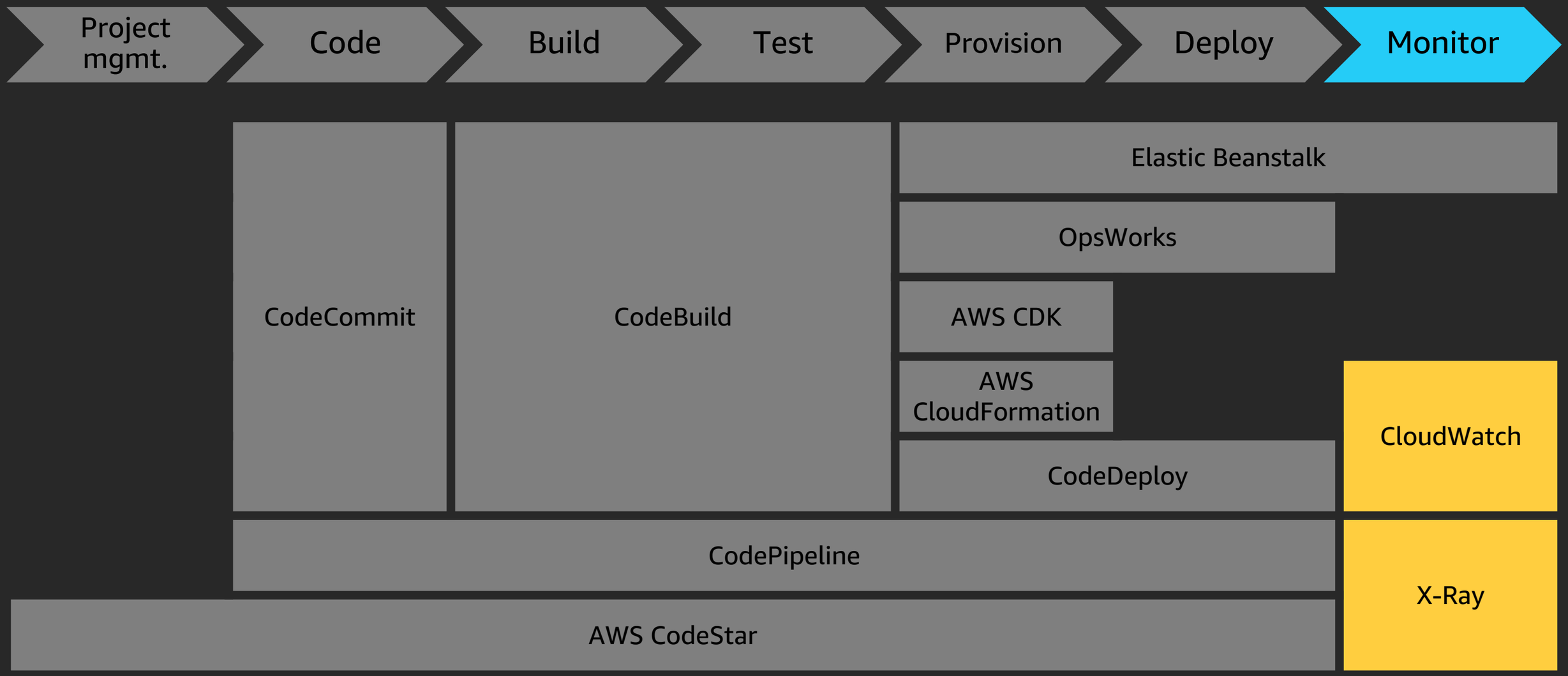
# Application lifecycle on AWS



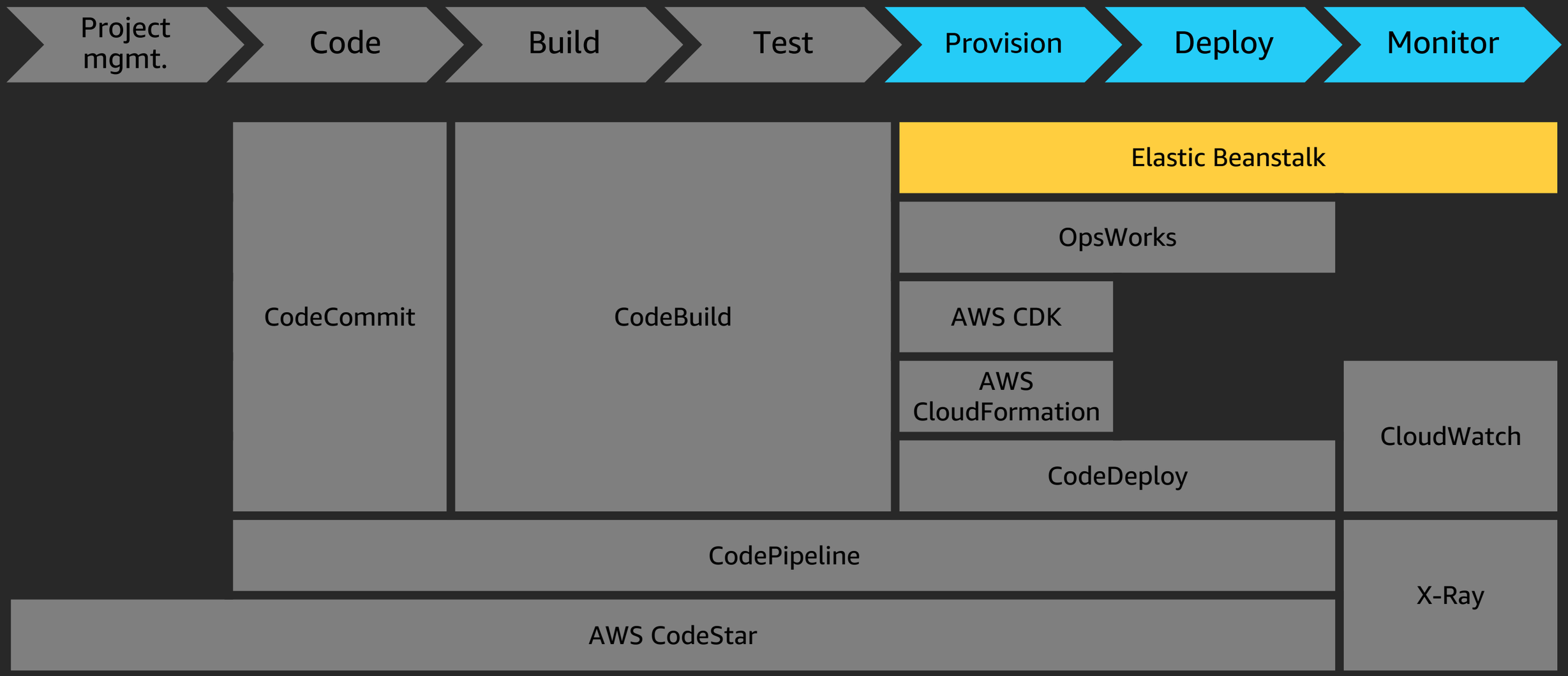
# Application lifecycle on AWS



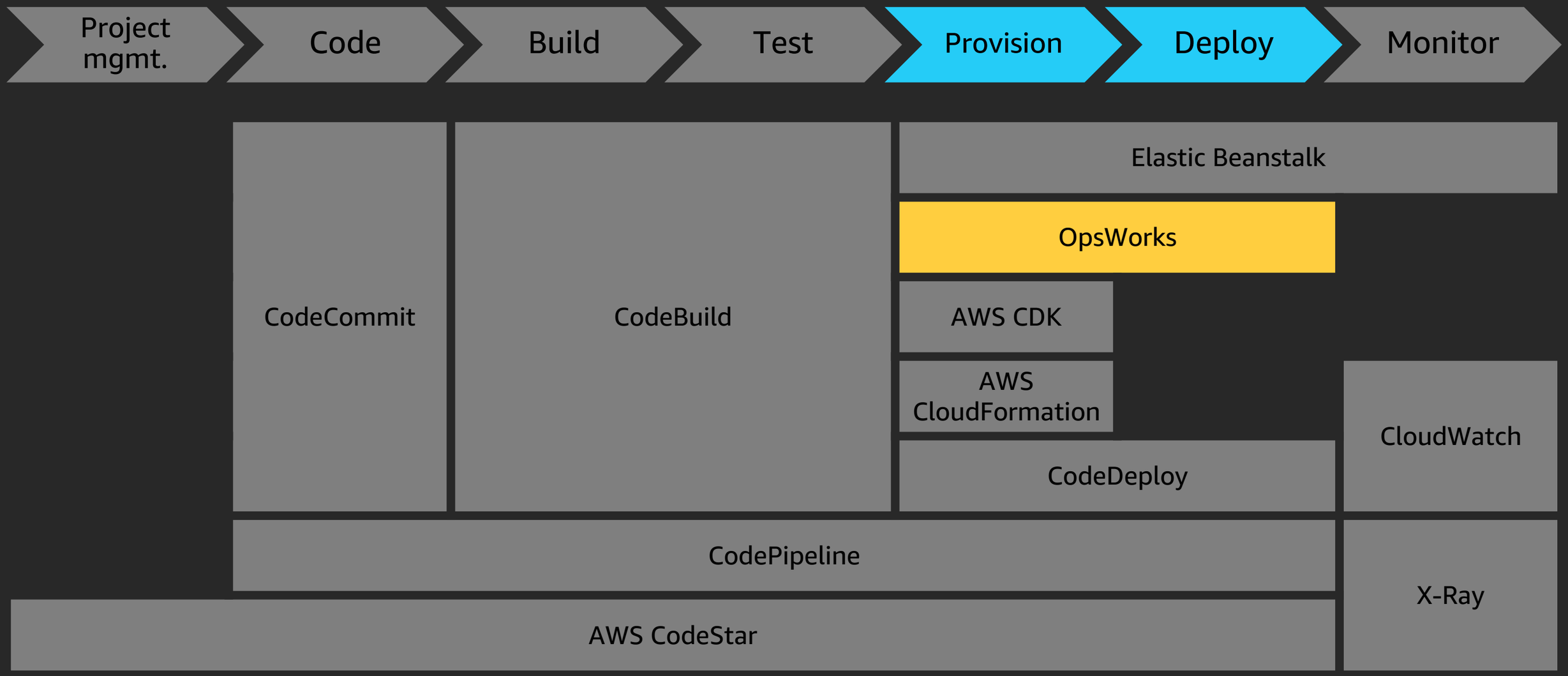
# Application lifecycle on AWS



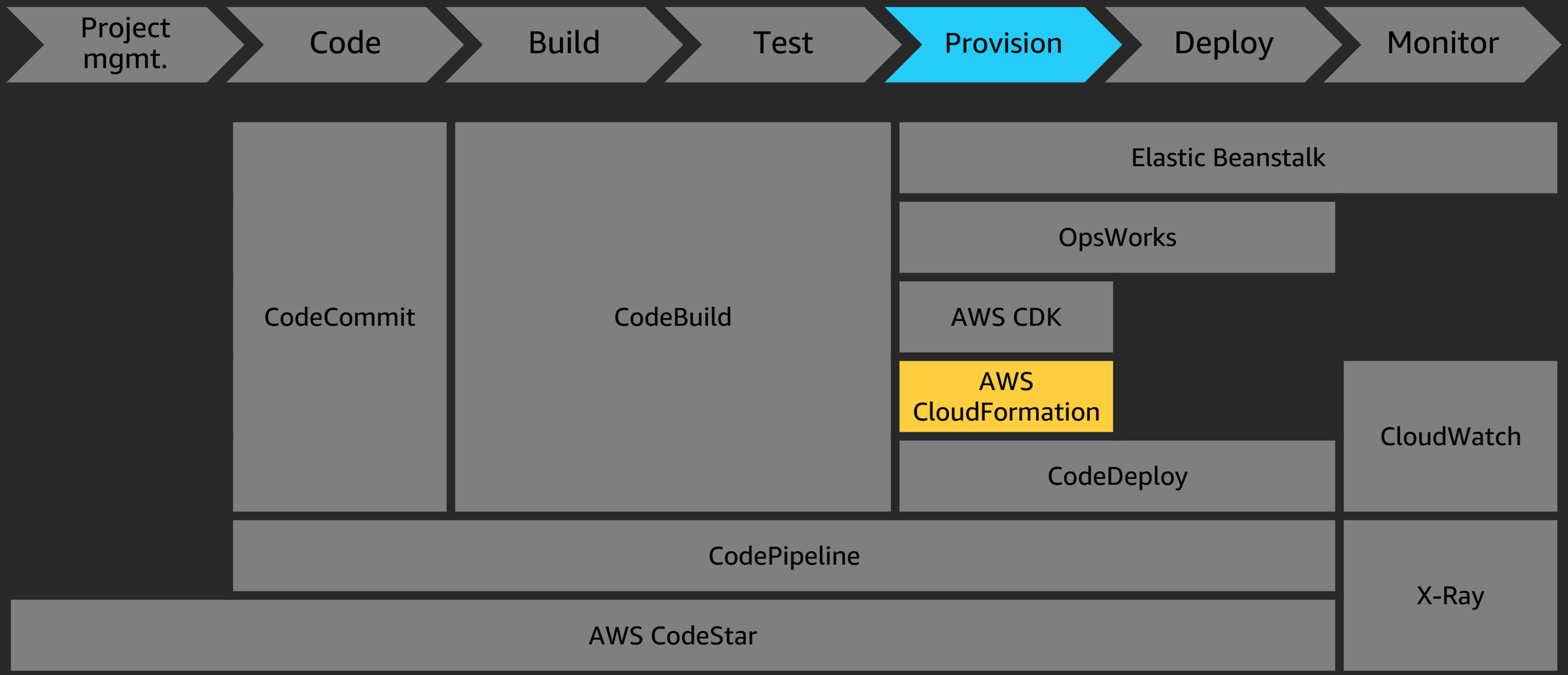
# Application lifecycle on AWS



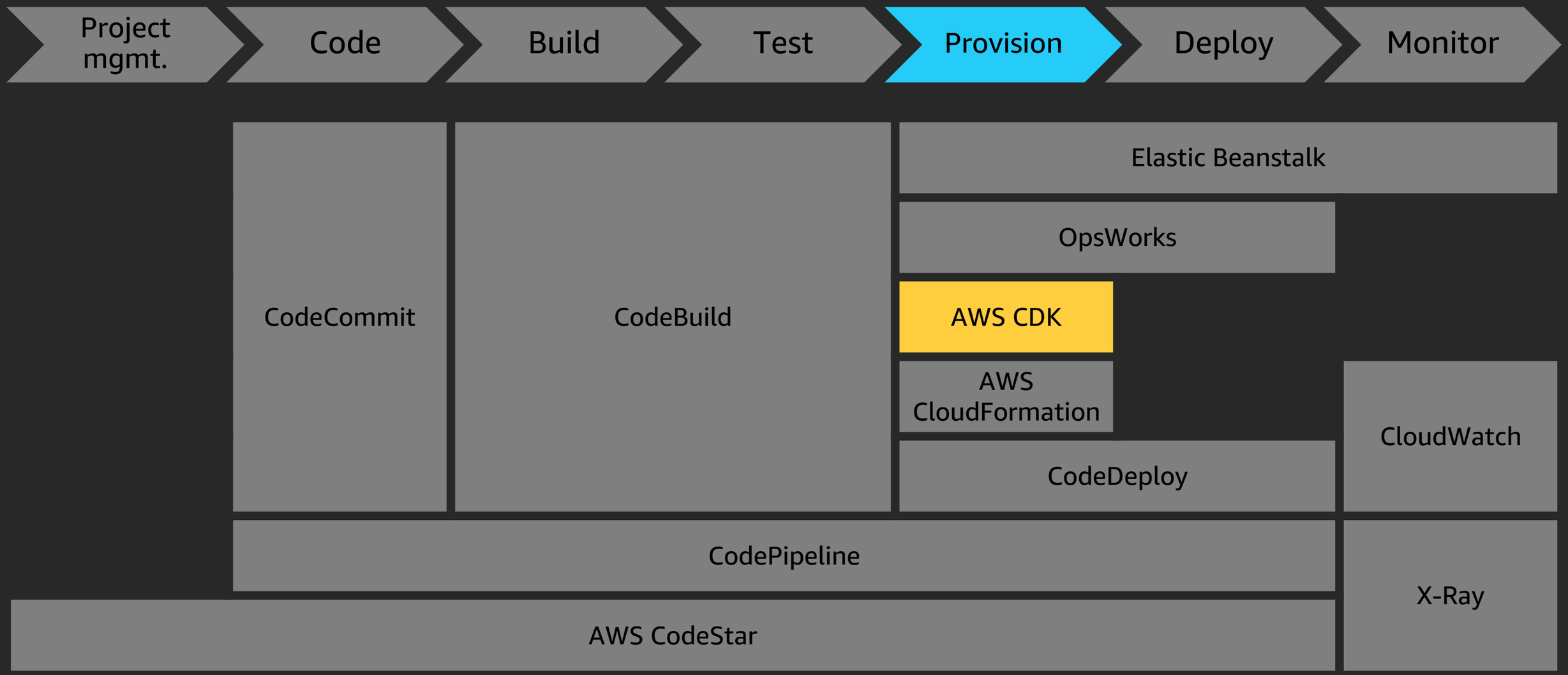
# Application lifecycle on AWS



# Application lifecycle on AWS



# Application lifecycle on AWS



# Infrastructure as Code (IaC)

# Infrastructure as Code (IaC)

Infrastructure definition files

No manual processes

Automation

Version control

# Advantages of IaC

Enable developers

Reduce cost

Execute faster

Reduce risk

# IaC approaches

Defines the desired state and the system executes what needs to happen to achieve that desired state

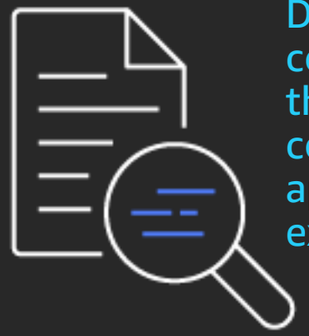
## Declarative—What?

What the event configuration should be

Defines the commands in specific order to achieve the desired state

## Imperative—How?

How the infrastructure should be changed to achieve the goal



Determines the correct desired state that does not impact co-dependent applications before executing

## Intelligent—Why?

Why configuration should be a certain way considering other dependent applications

# IaC Methods

Pulling configuration from the controlling server

## Pull

Puppet, Chef, OpsWorks

Controlling server pushing the configuration

## Push

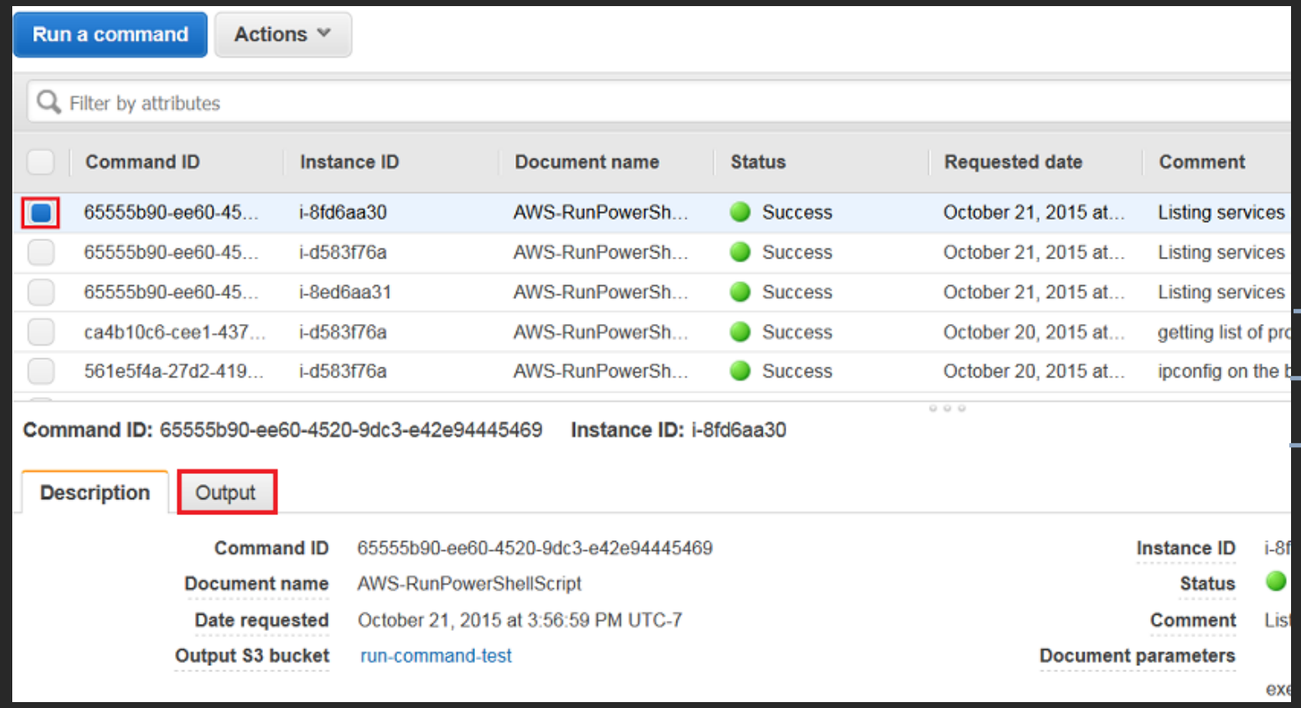
AWS CDK, AWS CloudFormation

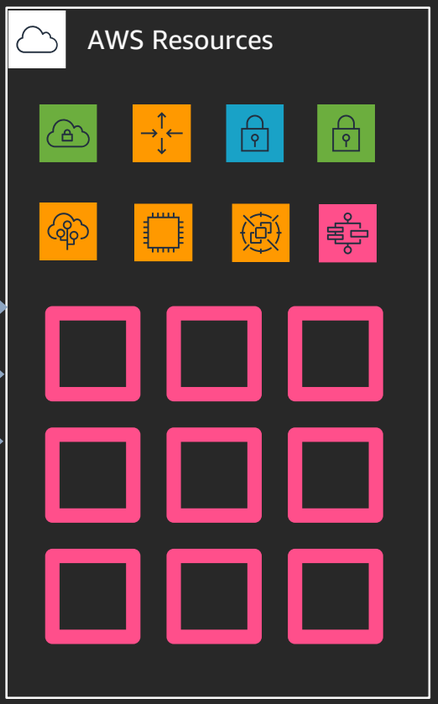
# AWS CDK

# Deployment journey on AWS

Manual

AWS Management Console



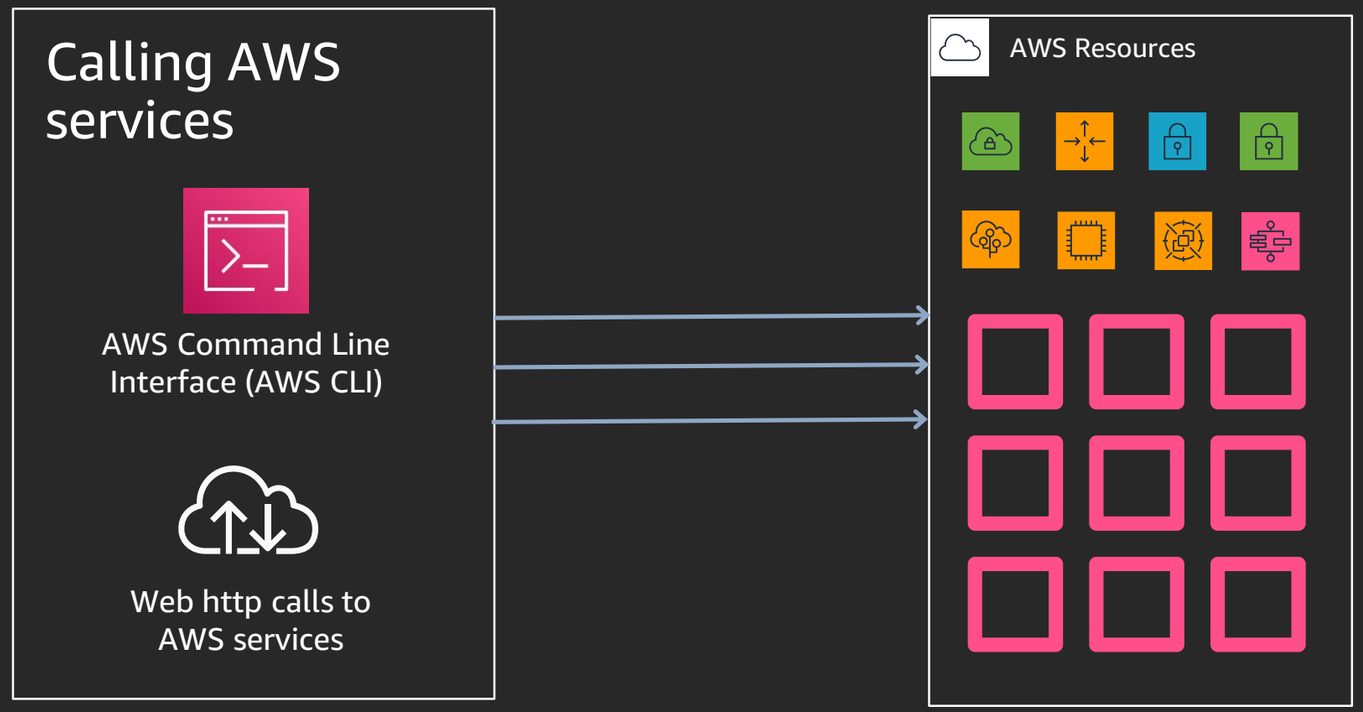



# Deployment journey on AWS

Manual → Scripted

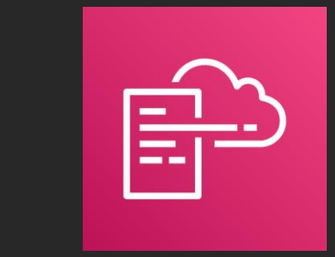
Scripting by calling AWS services API:

- Command line
- Web http calls
- Other 3rd party frameworks

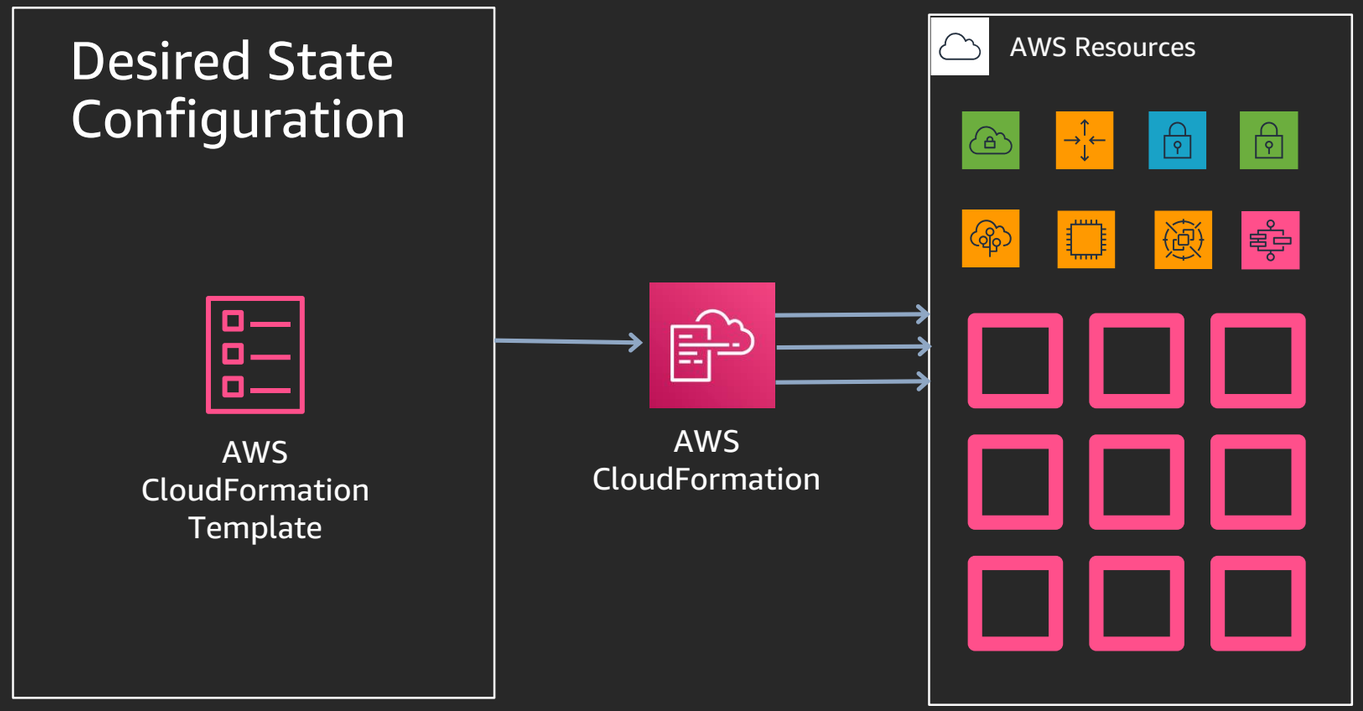


# Deployment journey on AWS

Manual | Scripted | Declarative



AWS CloudFormation



# Deployment journey on AWS

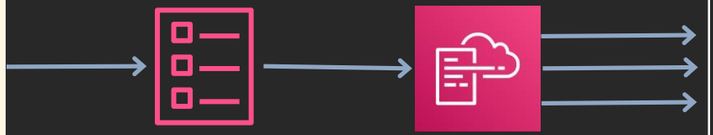
Manual → Scripted → Declarative → DOMs

Troposphere
SparkleFormation
GoFormation

And many more . . .

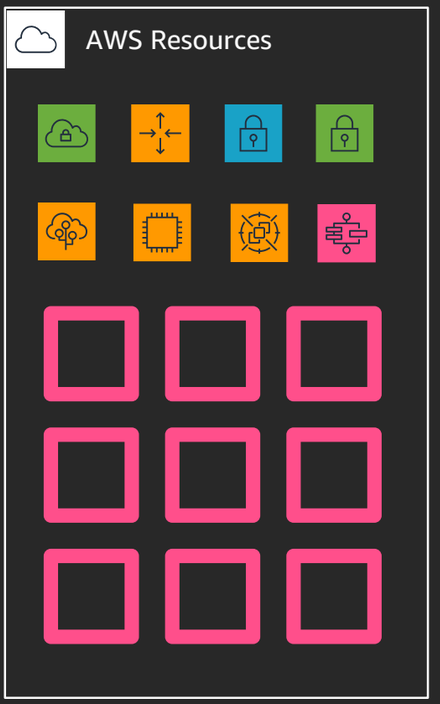
```python
from troposphere import Template
from troposphere.ec2 import VPC, Subnet, InternetGateway

t = Template()

VPC = t.add_resource(
    VPC(
        'VPC',
        CidrBlock='10.0.0.0/16',
        Tags=Tags(
            Application=ref_stack_id)))

subnet = t.add_resource(
    Subnet(
        'Subnet',
        CidrBlock='10.0.0.0/24',
        VpcId=Ref(VPC),
        Tags=Tags(
            Application=ref_stack_id)))
```

Troposphere *Python*
SparkleFormation *Ruby*
GoFormation *Go*
…



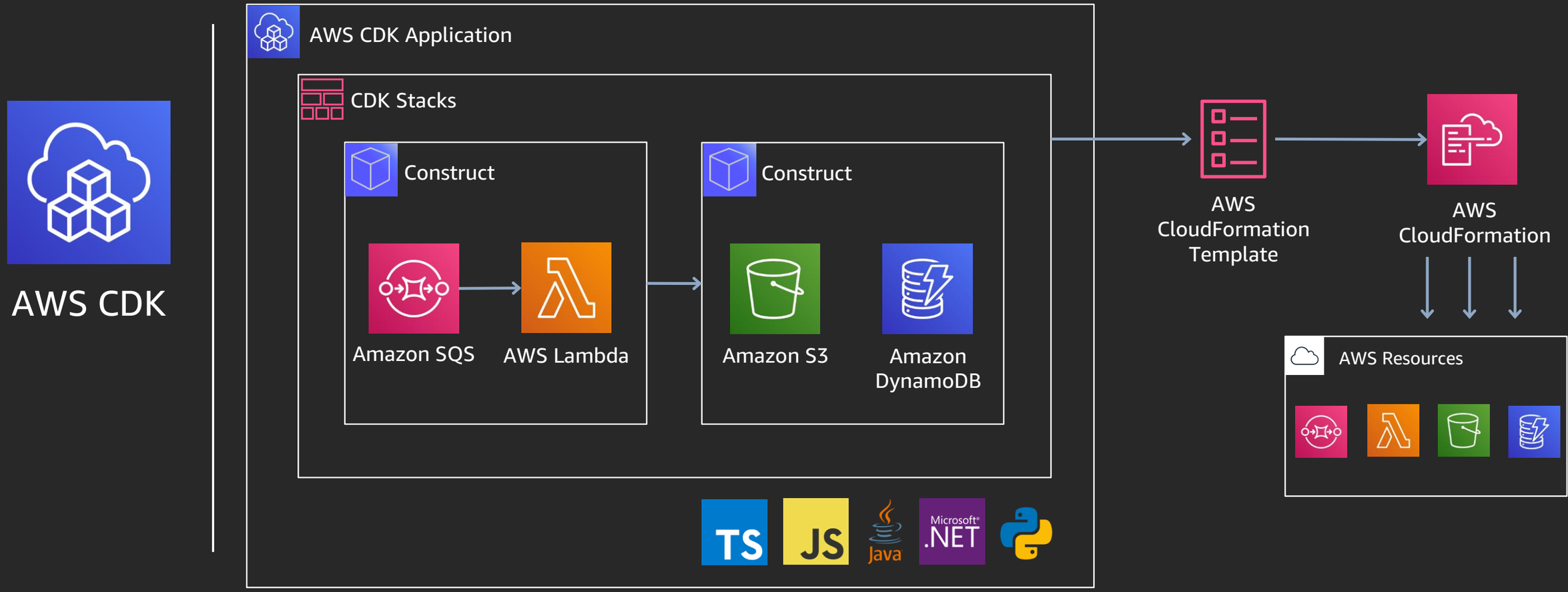
AWS CloudFormation Template

AWS CloudFormation



# Deployment journey on AWS

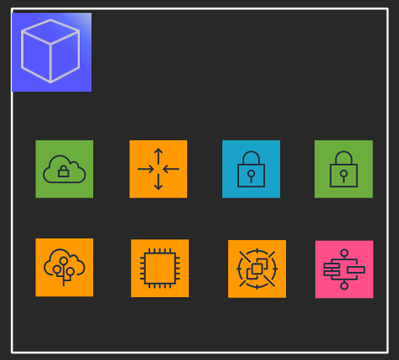
Manual → Scripted → Declarative → DOMs → **Componentized**



# AWS Cloud Development Kit (AWS CDK)

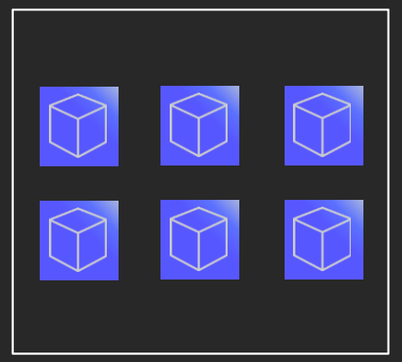

## High-level language

Treat infrastructure as objects
Logic statements
Object-oriented techniques
Organize in modules



## Constructs

Encapsulate complex architecture
Pre-defined configuration with best practices



## Predictable & Repeatable

Deployments can be predicted as is, as there is no manual intervention
Constructs can be shared, reused

# AWS CDK workshop

AWS re:Invent



aws

# CDK commands

cdk bootstrap - Deploys the CDK toolkit stack into an AWS environment

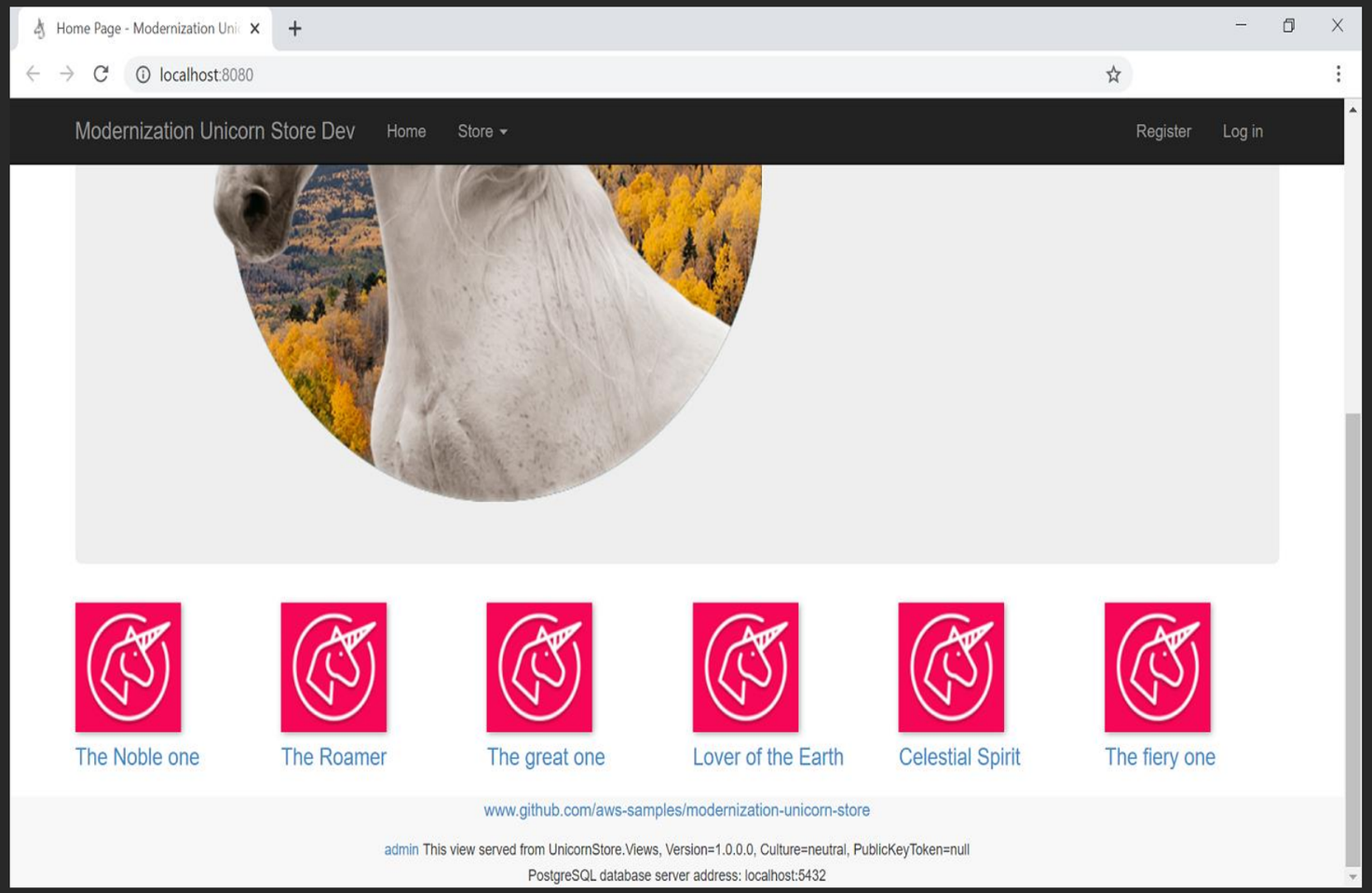
cdk init - Creates a new CDK project

cdk synth - Synthesizes and prints the AWS CloudFormation template for the stack

cdk deploy - Deploys the stack(s) into your AWS account

cdk destroy - Destroys the stack(s)

# Unicorn Store ASP.NET Core 3.0 MVC Application



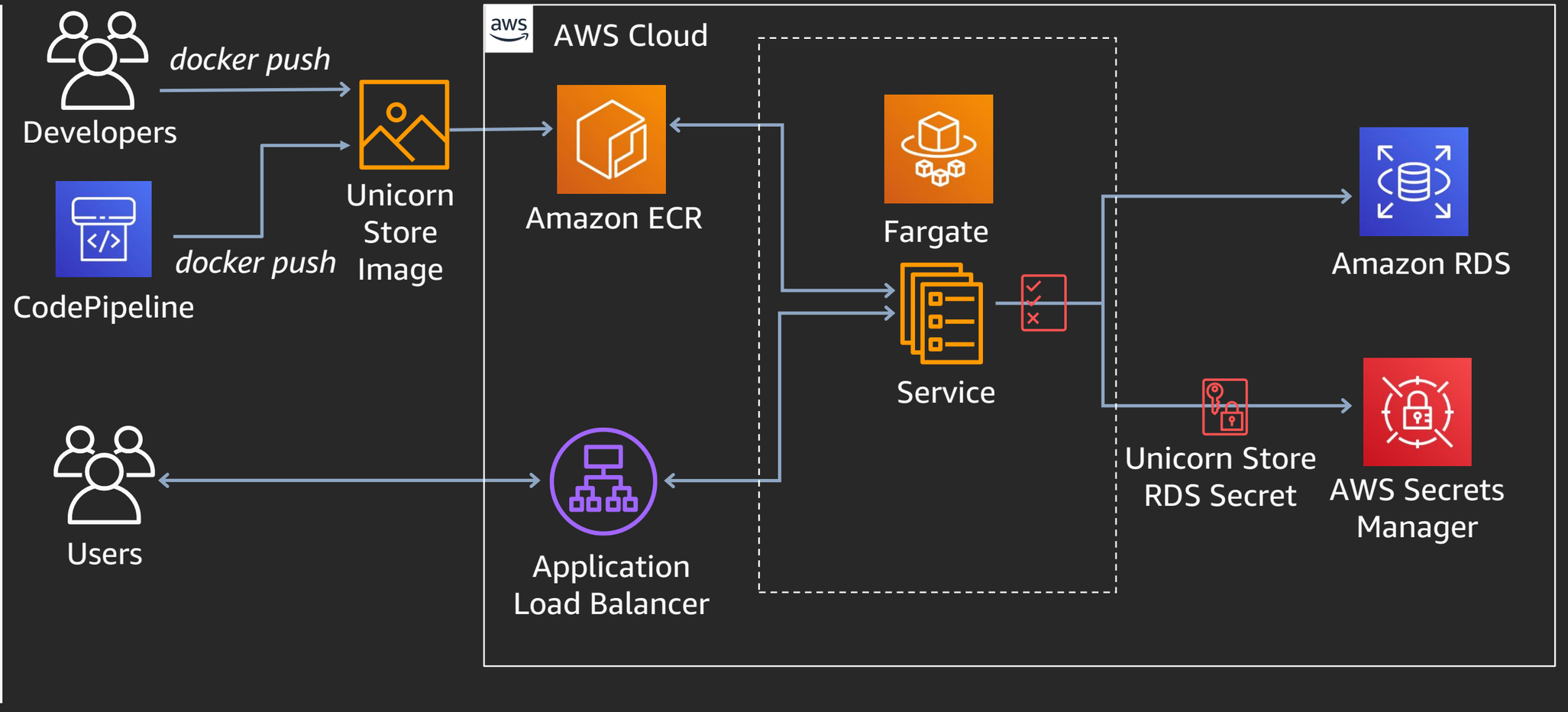

- Only open source dependencies
- Entity Framework Code First
- Cross-platform: runs in Linux containers
- Not aware of AWS infrastructure

# Unicorn Store Application architecture

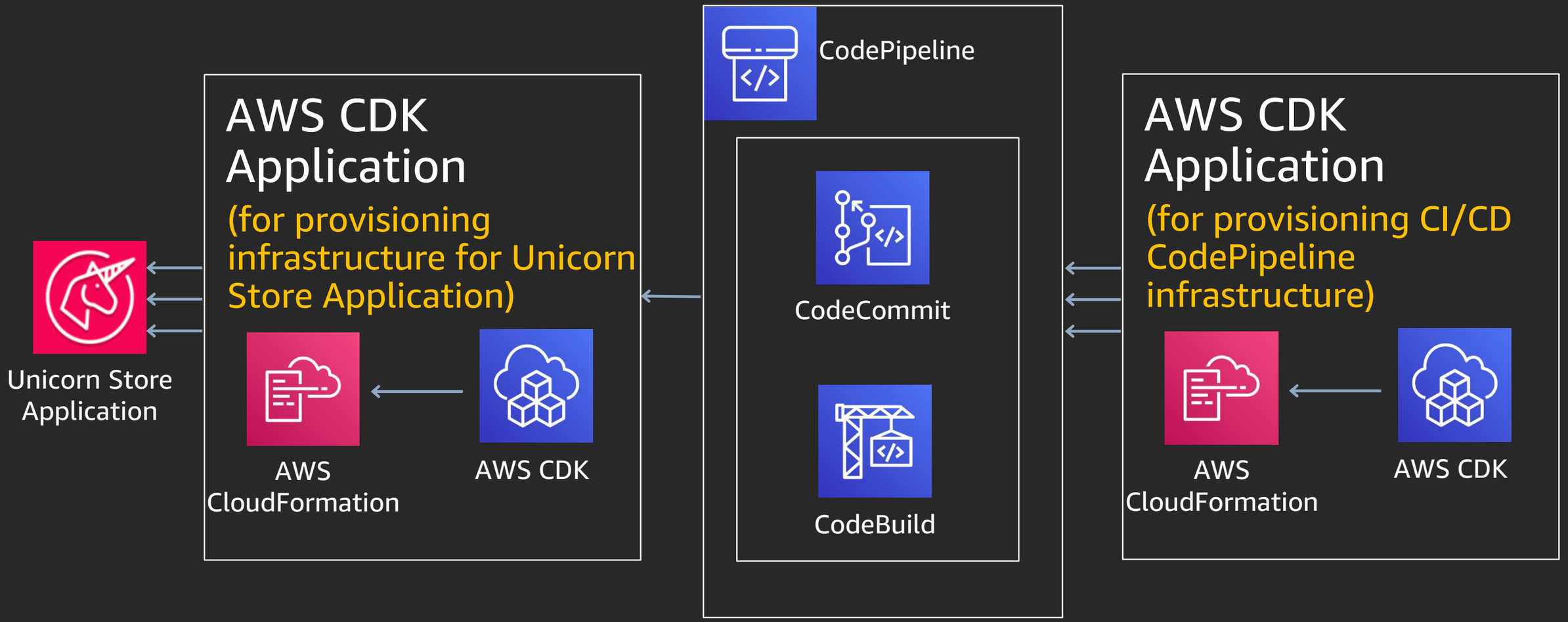
Unicorn Store Application

Application:
.NET core
Entity Framework

AWS services:
Amazon ECR
AWS Fargate
Amazon RDS
Amazon Aurora



# Unicorn Store Application deployment flow



# Database freedom

AWS CDK application

As all infrastructure components are treated imperatively like objects, based on the object type:

- Different docker image is generated
- Corresponding database is deployed


Unicorn Store Application

Microsoft SQL Server

Amazon RDS for SQL Server

PostgreSQL

Amazon Aurora PostgreSQL

MySQL

Amazon Aurora MySQL

# Accessing AWS Console and Development VM

- Go to https://dashboard.eventengine.run
- Please log out if you are still logged in from any previous session
- Enter hash from the printout and go to the AWS Console
- In AWS Web Console, go:
  Services | EC2 | Instances | <instance> | Actions | Connect | Download…
- Remote Desktop login: Administrator/Passw0rd (with digit 0)
- Use desktop icons to open Workshop Guide and Visual Studio solution

# Additional resources

- Repeat the workshop at home:
  https://tinyurl.com/github-dotnet-cdk

- YouTube:
  WIN310 – “Infrastructure as .NET with the AWS CDK” breakout session

# Thank you!

**Rahul Chugh**
@imRahulChugh

**Vlad Hrybok**
@VladHrybok

# Please complete the session survey in the mobile app.