Ran The Builder
Build Serverless Services

LEVEL UP YOUR CI/CD WITH AWS SMART FEATURE FLAGS

RAN ISENBERG, PRINCIPAL SOFTWARE ARCHITECT

CYBERARK®


aws

INTRODUCTION

- Principal Software Architect @CyberArk
- AWS Community Builder
- Owner & Blogger @RanTheBuilder.Cloud

AGENDA

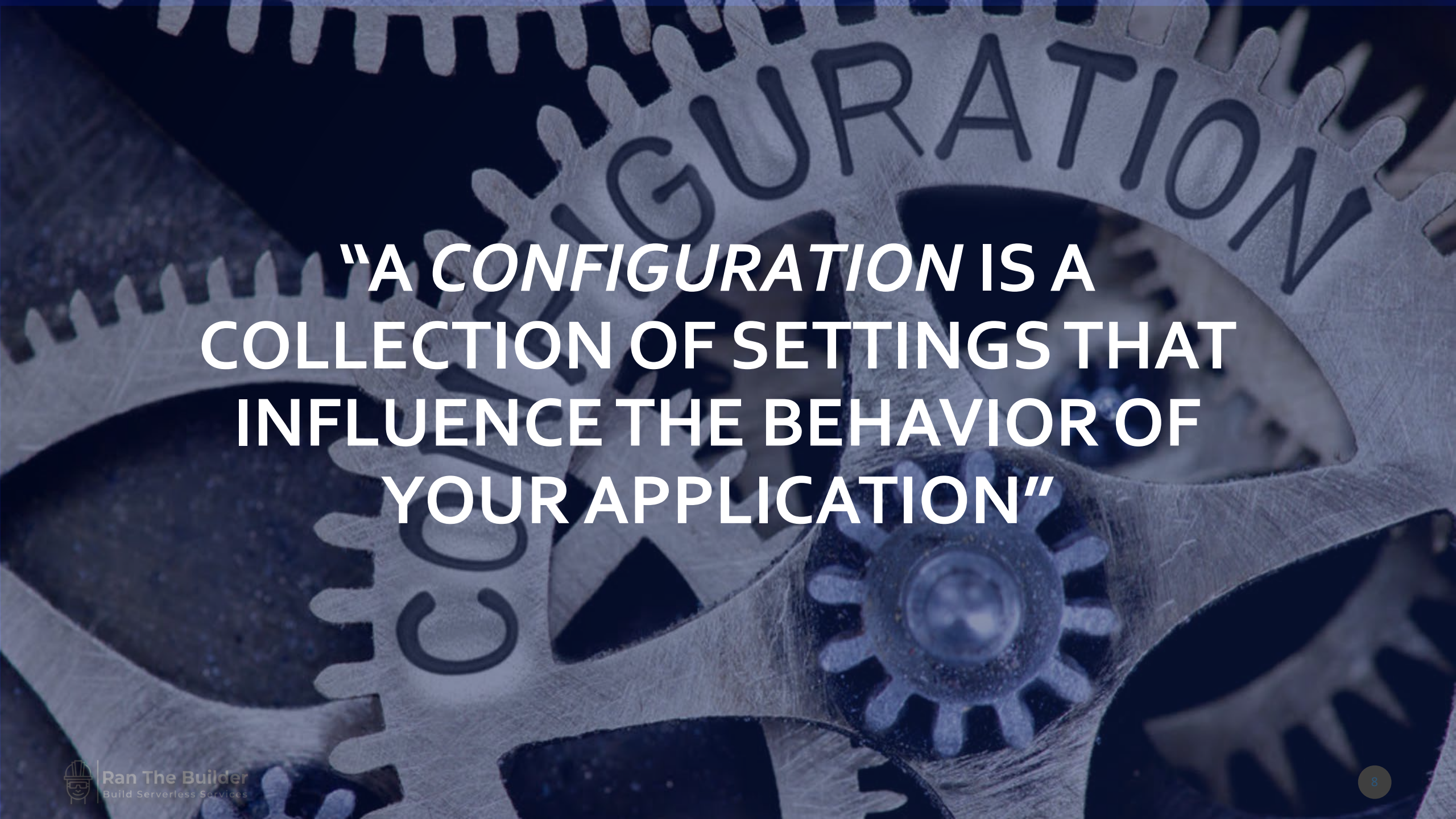
Requirements	• Functional • Non-functional
Configuration Types	• Static vs. Dynamic
Solution	• AWS AppConfig • AWS Lambda Powertools smart feature flags
Best Practices	• All CI/CD stages • A/B testing and canary deployment examples

REQUIREMENTS

Functional & Non-Functional

REQUIREMENTS

Deployment Types	• Gradual deployment of features • A/B testing
Act Quickly	• Automatic rollback • Disable features ASAP
AWS Solution	• Supports Lambda functions/containers • FedRamp High certification
Non-Functional	• Easy to use & integrate • Self managed & resilient

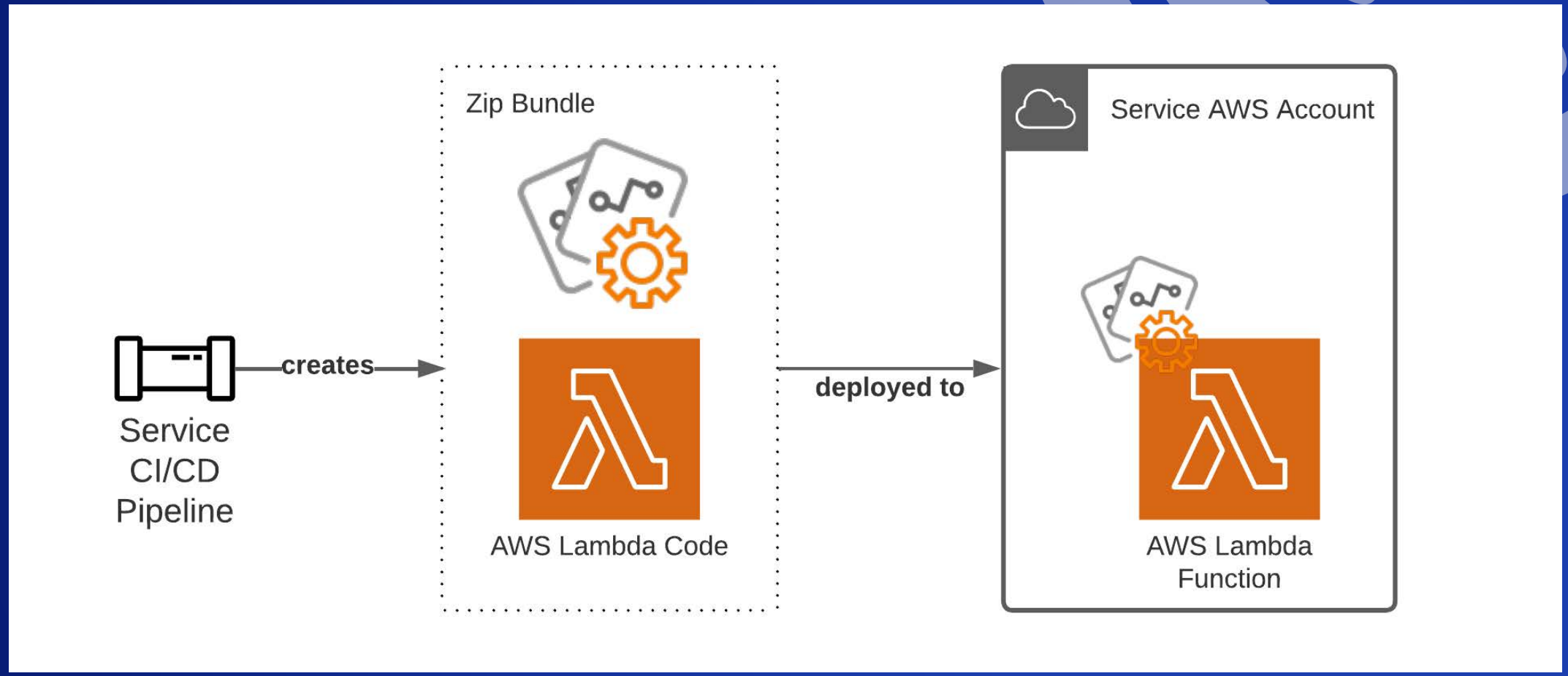

"A *CONFIGURATION* IS A COLLECTION OF SETTINGS THAT INFLUENCE THE BEHAVIOR OF YOUR APPLICATION"

Naïve Feature Flags Impl.

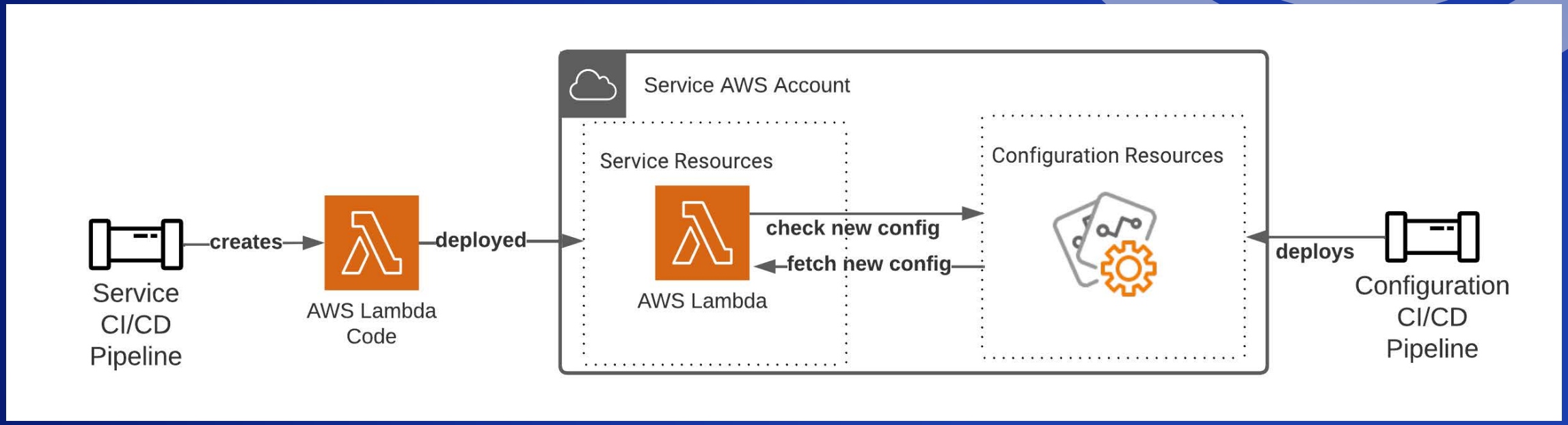
```
def my_func():
  feature_flag: bool = evaluate_feature_flag()
  if feature_flag:
    handle_new_feature_logic()
  else:
    handle_regular_logic()
```

CONFIGURATION TYPES

STATIC


Zip Bundle
Service AWS Account
creates
deployed to
Service
CI/CD
Pipeline
AWS Lambda Code
AWS Lambda
Function

DYNAMIC


Service AWS Account
Service Resources
Configuration Resources
creates
deployed
check new config
fetch new config
deploys
Service
CI/CD
Pipeline
AWS Lambda
Code
AWS Lambda
Configuration
CI/CD
Pipeline

Static vs. Dynamic

- Service reads configuration from bundled resources
- Change requires service CI/CD pipeline redeployment
- Slow changes in service behavior
- Easier to manage

- Service reads configuration from an external source in runtime
- Changes require configuration CI/CD pipeline
- **Quick changes in service behavior**
- Harder to manage, increased complexity

SOLUTION

SOLUTION OVERVIEW

- Develop
  - Configuration JSON file
- Store & Deploy
  - AWS AppConfig
  - Dedicated CI/CD Pipeline
- Evaluate
  - AWS Lambda Powertools
  - Feature Flags SDK

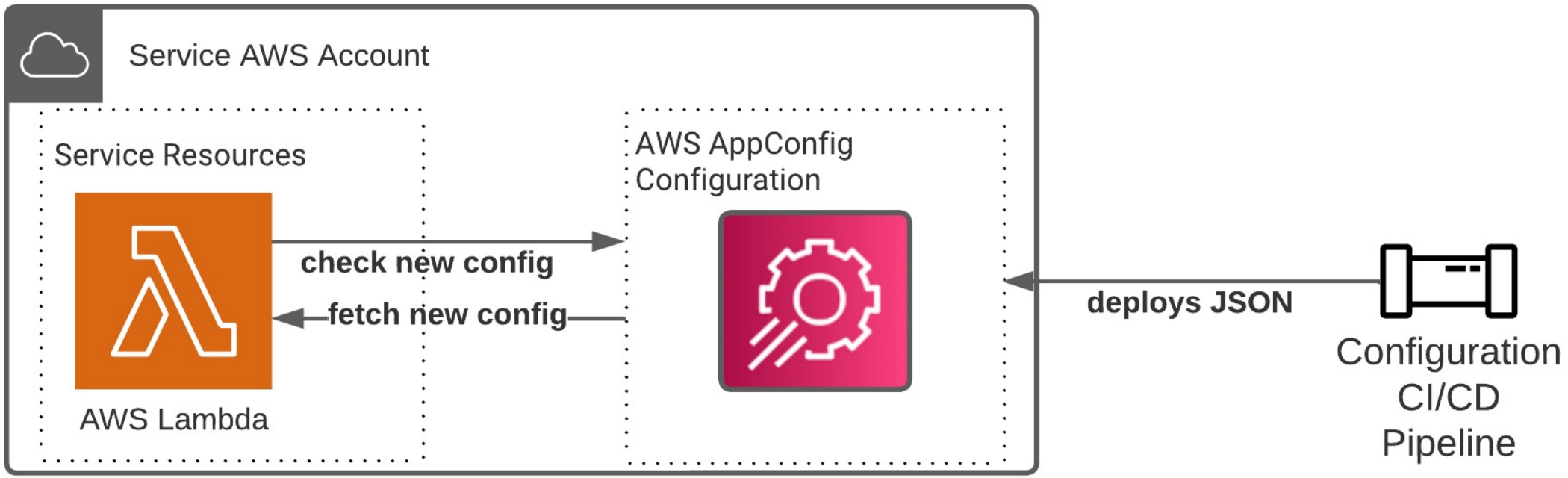
Develop

- JSON Configuration File

```
{
    "premium_features": {
        "default": false,
    }
}
```

STORE & DEPLOY

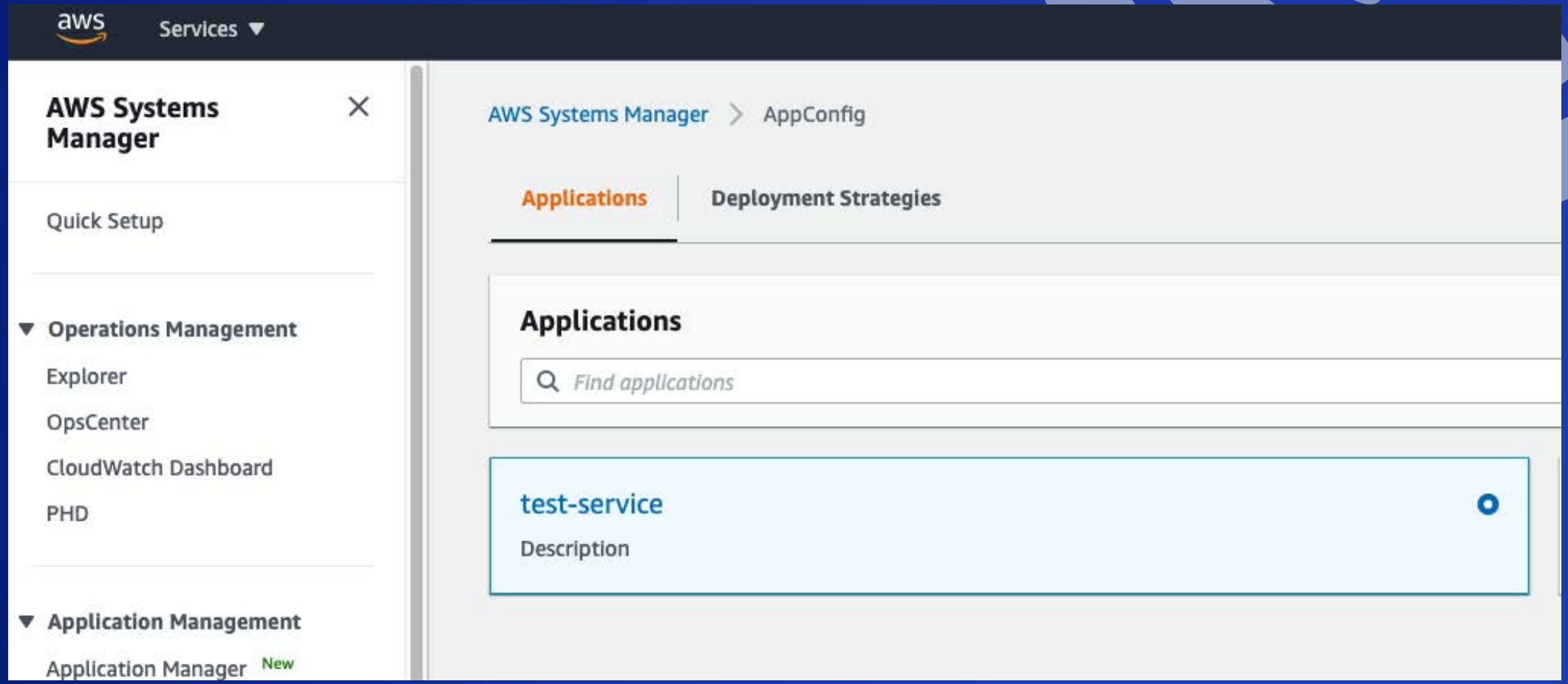
AWS APPCONFIG CI/CD PIPELINE



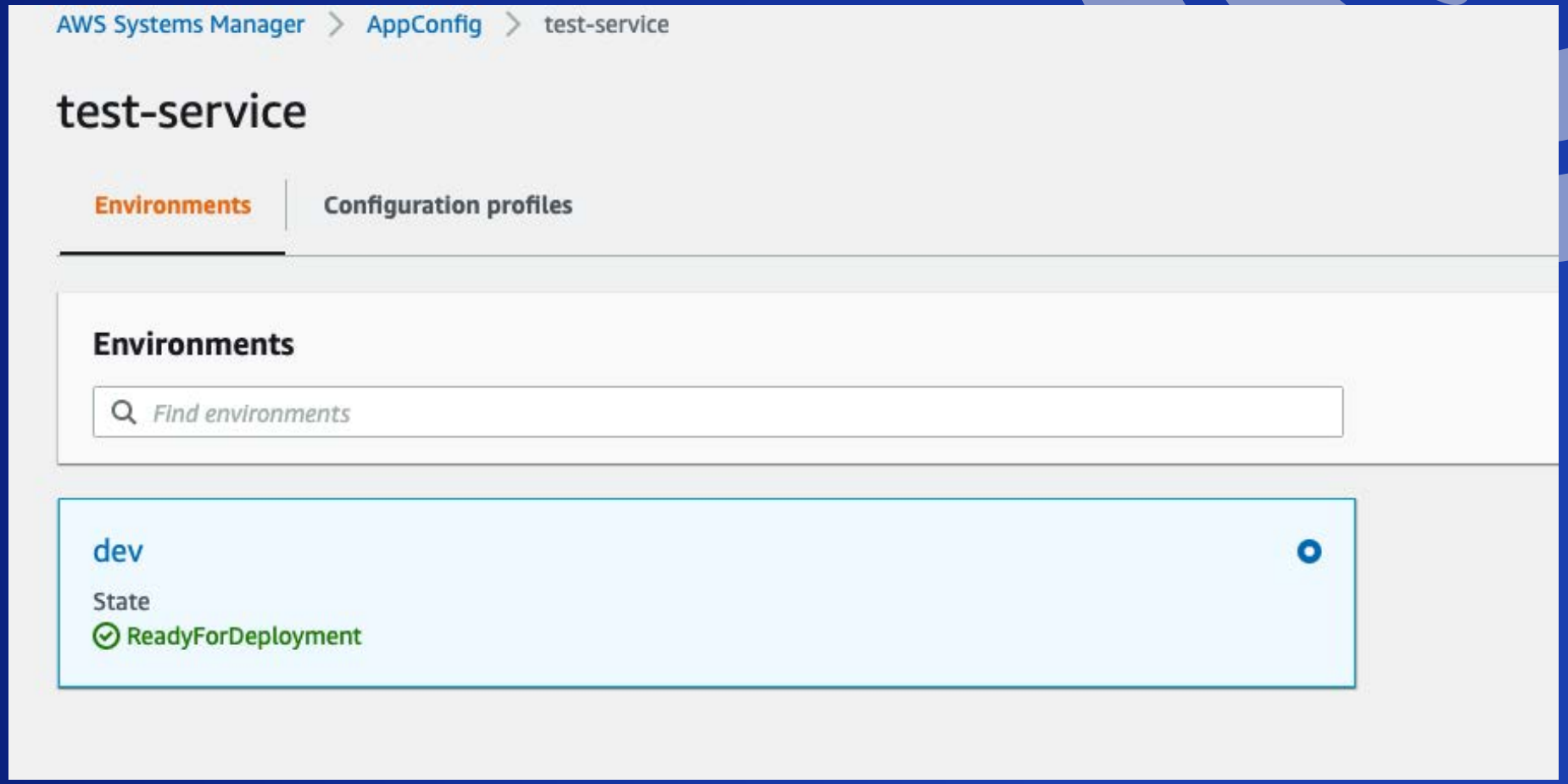
AWS APPCONFIG

- AWS service, no 3rd Party Integration
- FedRAMP High certified
- Fully managed (backups, high availability)
- Validate JSON schema
- Deployment strategies (canary deployment)
- Monitor & rollback (versions)

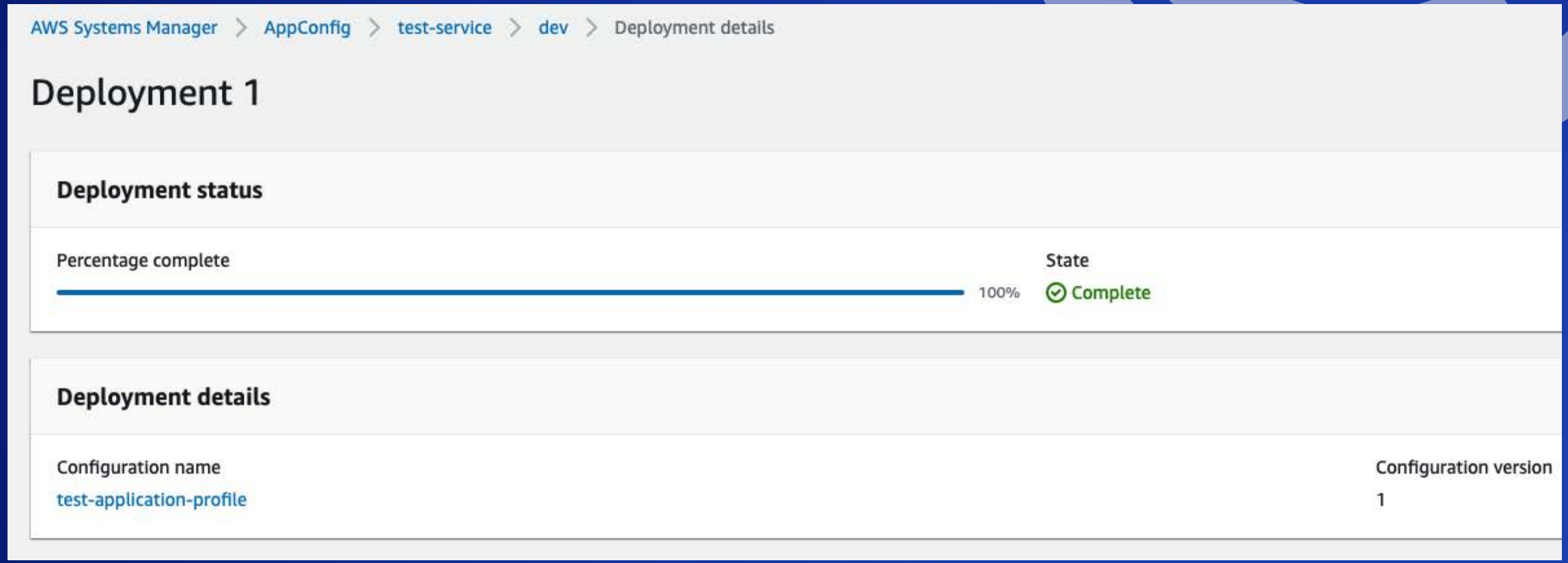
APPLICATION



ENVIRONMENT



DEPLOYED CONFIGURATION



EVALUATE

AWS Lambda Powertools

AWS Lambda Powertools (Python)

- AWS Labs GitHub repository
- Over 1400 stars, Over 1 million downloads/month
- Defines best practices for AWS Lambda

Feature Flags Utility

- Fetch configuration from AppConfig, store in cache
- Evaluate feature flags value
- Regular & Smart feature flags rule engine
- Not just for Lambda functions

Sample Use Case – Regular Flags

```
{
    ”ten_percent_off_campaign": {
            "default": true,
      }
  }
```

REGULAR FEATURE FLAG

```python
from aws_lambda_powertools.utilities.feature_flags import FeatureFlags, AppConfigStore

app_config = AppConfigStore(
    environment="dev",
    application="product-catalogue",
    name="features"
)

feature_flags = FeatureFlags(store=app_config)

def lambda_handler(event, context):
    apply_discount: bool = feature_flags.evaluate(name="ten_percent_off_campaign",
                                                  default=False)

    if apply_discount:
        # apply 10% discount to product
        ...
```

SMART FEATURE FLAGS

- Simple rule engine SDK
- Evaluated in runtime
- Flags change value according to input context
- Generic context & action rich language
- A/B testing enabler

Sample Configuration

Input Event

```
{
    "username": "lessa",
    "tier": "premium",
    "basked_id": "random_id"
}
```

JSON Config

```
{
    "premium_features": {
        "default": false,
        "rules": {
            "customer tier equals premium": {
                "when_match": true,
                "conditions": [
                    {
                        "action": "EQUALS",
                        "key": "tier",
                        "value": "premium"
                    }
                ]
            }
        }
    },
    "ten_percent_off_campaign": {
        "default": false
    }
}
```

SMART FEATURE FLAGS

```python
from aws_lambda_powertools.utilities.feature_flags import FeatureFlags, AppConfigStore

app_config = AppConfigStore(
    environment="dev",
    application="product-catalogue",
    name="features"
)

feature_flags = FeatureFlags(store=app_config)

def lambda_handler(event, context):
    # Get customer's tier from incoming request
    ctx = { "tier": event.get("tier", "standard") }

    # Evaluate whether customer's tier has access to premium features
    # based on `has_premium_features` rules
    has_premium_features: bool = feature_flags.evaluate(name="premium_features",
                                                        context=ctx, default=False)
    if has_premium_features:
        # enable premium features
        ...
```

Actions

- EQUALS
- NOT_EQUALS
- KEY_GREATER_THAN_VALUE
- STARTSWITH
- KEY_IN_VALUE
- And many more

NON-BOOLEAN FEATURE FLAGS

```
    "non_boolean_premium_feature": {
        "default": [],
        "rules": {
            "customer tier equals premium": {
                "when_match": ["remove_limits", "remove_ads"],
                "conditions": [
                    {
                        "action": "EQUALS",
                        "key": "tier",
                        "value": "premium"
                    }
                ]
```

SAMPLE RULES

- Enable a feature for a specific:
  - Customer
  - Users of a customer (admin etc.)
- Apply discount for specific types of products
- Offer free shipping if total cost is higher than X
- Endless possibilities

A/B Testing

- Smart Feature Flags framework of A/B testing
- Different user experience for different users with a single configuration

CACHE

- Change? Run configuration CI/CD pipeline
- Cache expires -> behavior change

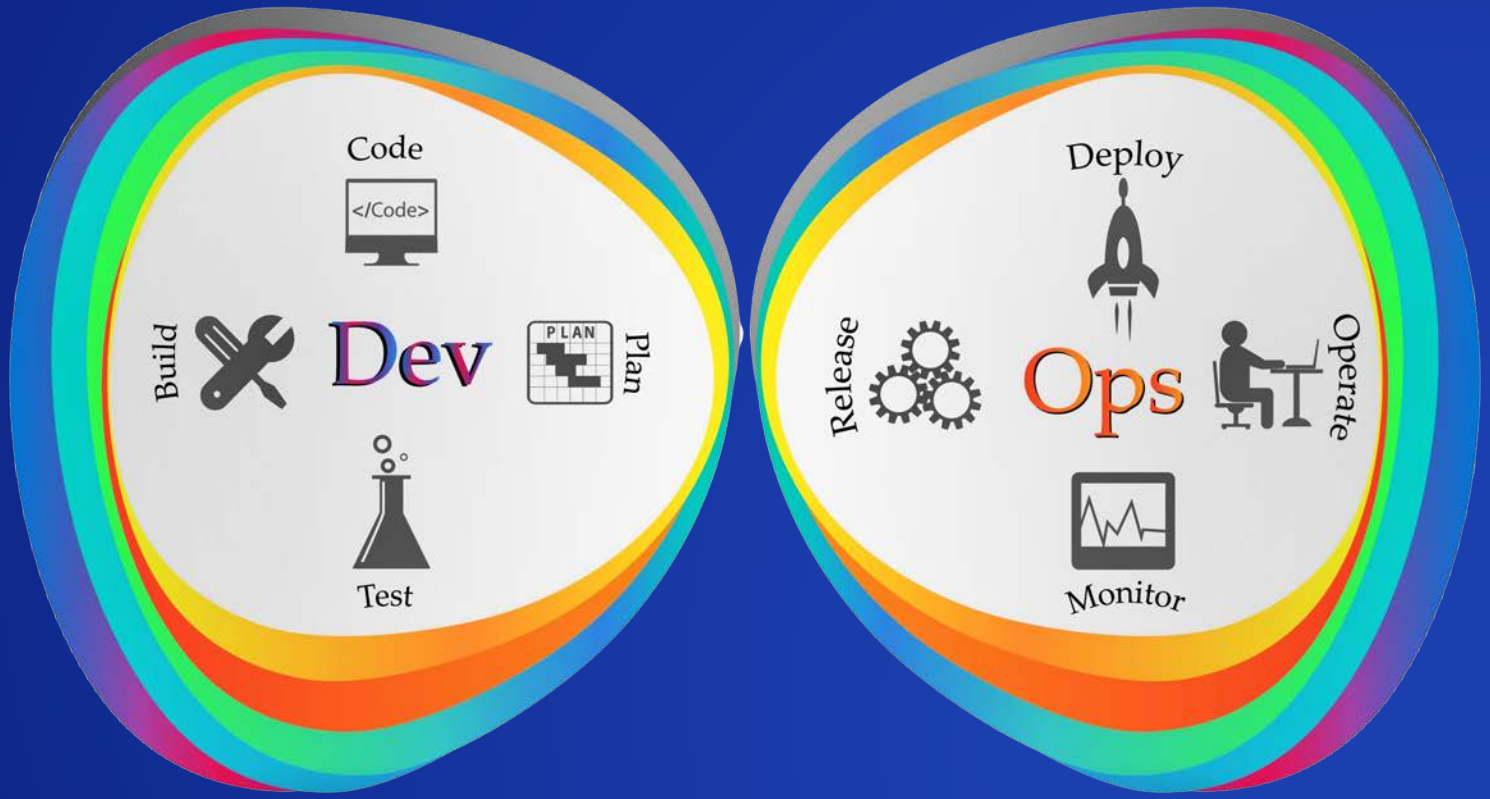
```python
from aws_lambda_powertools.utilities.feature_flags import FeatureFlags, AppConfigStore

app_config = AppConfigStore(
    environment="dev",
    application="product-catalogue",
    name="features",
    max_age=300
)
```

COMING SOON

- Time based rules:
  - Enable at specific time
  - Enable for a specific duration
  - Enable/disable during specific days

FEATURE FLAGS BEST PRACTICES



Plan, Code & Build

- Ownership of dev team from start to end
- Plan config JSON rules
- Write code that evaluates it
- Disabled in production, enabled in dev/test accounts

Test

- Mock configuration in local IDE tests
- Mock feature is disabled
  - Assert feature handing code does NOT run
- Mock feature is enabled
  - Assert feature handling code runs
  - Side effects are valid

Release & Deploy

- Once feature is stable in non-production environment
- Deployment strategy to production
  - Canary (AppConfig.Canary10Percent20Minutes)
  - All at once option
- Auto revert with CloudWatch alarms

Monitor & Operate

- Error? Disable the feature flag ASAP
- Restart configuration CI/CD process
  - Update tests - add missing use cases
  - Deploy and re-release
- Conduct retro meeting
  - Identify overlooked use cases in tests

Retire

- Why?
  - Reduce code complexity
  - Easier to maintain
  - Better visibility on overall flags
- How?
  - Meeting once a month
  - Remove & deploy configuration CI/CD pipeline

Retire – Contd.

When?

- Feature enabled to 100% of customers for ‘X’ weeks.
- Feature is stable for ‘X’.
- Customer feedback is positive, and there are no open issues.
- The code surrounding the feature is not expected to undergo any refactors/additions

Summary

- We created feature flags (smart & regular)
- Deployed to AWS AppConfig
- Evaluated in runtime with AWS Lambda Powertools.
- We created canary deployments
- We conducted A/B testing
- We learnt feature flags best practices

Ran The Builder
Build Serverless Services

CYBERARK®

THANK YOU!

@RANISENBERG

HTTPS://WWW.RANTHEBUILDER.CLOUD

@ISENBERGRAN