RDK Documentation

Release 1.0

Michael Borchert

Nov 06, 2019

Contents:

CHAPTER 1

Introduction

Rule Development Kit - Version 2 This tool should be considered in "Open Beta". We would greatly appreciate feedback and bug reports either as github issues or emails to rdk-maintainers@amazon.com!

The RDK is designed to support a "Compliance-as-Code" workflow that is intuitive and productive. It abstracts away much of the undifferentiated heavy lifting associated with deploying AWS Config rules backed by custom lambda functions, and provides a streamlined develop-deploy-monitor iterative process.

CHAPTER 2

Getting Started

Let's get started using the RDK!

2.1 Prerequisites

RDK uses python 2.7/3.6+. You will need to have an AWS account and sufficient permissions to manage the Config service, and to create and manage S3 Buckets, Roles, and Lambda Functions. An AWS IAM Policy Document that describes the minimum necessary permissions can be found here on github.

Under the hood, rdk uses boto3 to make API calls to AWS, so you can set your credentials any way that boto3 recognizes (options 3 through 8 in the boto docs here) or pass them in with the command-line parameters –profile, –region, –access-key-id, or –secret-access-key

2.2 Installation

If you just want to use the RDK, go ahead and install it using pip:

```
$ pip install rdk
```

Alternately, if you want to see the code and/or contribute you can clone the git repo , and then from the repo directory use pip to install the package. Use the '-e' flag to generate symlinks so that any edits you make will be reflected when you run the installed package.

If you are going to author your Lambda functions using Java you will need to have Java 8 and gradle installed. If you are going to author your Lambda functions in C# you will need to have the dotnet CLI and the .NET Core Runtime 1.08 installed.

```
$ pip install -e .
```

To make sure the rdk is installed correctly, running the package from the command line without any arguments should display help information.

```
$ rdk
usage: rdk [-h] [-p PROFILE] [-k ACCESS_KEY] [-s SECRET_ACCESS_KEY]
         [-r REGION]
         <command> ...
rdk: error: the following arguments are required: <command>, <command arguments>
```

2.3 Usage

2.3.1 Configure your env

To use the RDK, it's recommended to create a directory that will be your working directory. This should be committed to a source code repo, and ideally created as a python virtualenv. In that directory, run the `init` command to set up your AWS Config environment.

```
$ rdk init
Running init!
Creating Config bucket config-bucket-780784666283
Creating IAM role config-role
Waiting for IAM role to propagate
Config Service is ON
Config setup complete.
Creating Code bucket config-rule-code-bucket-780784666283ap-southeast-1
```

Running `init` subsequent times will validate your AWS Config setup and re-create any S3 buckets or IAM resources that are needed.

2.3.2 Create Rules

In your working directory, use the `create` command to start creating a new custom rule. You must specify the runtime for the lambda function that will back the Rule, and you can also specify a resource type (or comma-separated list of types) that the Rule will evaluate or a maximum frequency for a periodic rule. This will add a new directory for the rule and populate it with several files, including a skeleton of your Lambda code.

```
$ rdk create MyRule --runtime python3.7 --resource-types AWS::EC2::Instance --input-
↪parameters '{"desiredInstanceType":"t2.micro"}'
Running create!
Local Rule files created.
```

On Windows it is necessary to escape the double-quotes when specifying input parameters, so the *–input-parameters* argument would instead look something like this:

```
'{\"desiredInstanceType\":\"t2.micro\"}'
```

Note that you can create rules that use EITHER resource-types OR maximum-frequency, but not both. We have found that rules that try to be both event-triggered as well as periodic wind up being very complicated and so we do not recommend it as a best practice.

2.3.3 Edit Rules Locally

Once you have created the rule, edit the python file in your rule directory (in the above example it would be `MyRule/MyRule.py`, but may be deeper into the rule directory tree depending on your chosen Lambda runtime) to add

whatever logic your Rule requires in the `evaluate_compliance` function. You will have access to the CI that was sent by Config, as well as any parameters configured for the Config Rule. Your function should return either a simple compliance status (one of `COMPLIANT`, `NONCOMPLIANT`, or `NOT_APPLICABLE`), or if you're using the python or node runtimes you can return a JSON object with multiple evaluation responses that the RDK will send back to AWS Config. An example would look like:

```
for sg in response['SecurityGroups']:
      evaluations.append(
      {
              'ComplianceResourceType': 'AWS::EC2::SecurityGroup',
              'ComplianceResourceId': sg['GroupId'],
              'ComplianceType': 'COMPLIANT',
              'Annotation': 'This is an important note.',
              'OrderingTimestamp': str(datetime.datetime.now())
      })


  return evaluations
```

This is necessary for periodic rules that are not triggered by any CI change (which means the CI that is passed in will be null), and also for attaching annotations to your evaluation results.

If you want to see what the JSON structure of a CI looks like for creating your logic, you can use

```
$ rdk sample-ci <Resource Type>
```

to output a formatted JSON document.

2.3.4 Write and Run Unit Tests

If you are writing Config Rules using either of the Python runtimes there will be a <rule name>_test.py file deployed along with your Lambda function skeleton. This can be used to write unit tests according to the standard Python unittest framework (documented here: https://docs.python.org/3/library/unittest.html), which can be run using the *test-local* rdk command:

```
$ rdk test-local MyTestRule
Running local test!
Testing MyTestRule
Looking for tests in /Users/mborch/Code/rdk-dev/MyTestRule

----------------------------------------------------------------------

Ran 0 tests in 0.000s

OK
<unittest.runner.TextTestResult run=0 errors=0 failures=0>
```

The test file includes setup for the MagicMock library that can be used to stub boto3 API calls if your rule logic will involve making API calls to gather additional information about your AWS environment. For some tips on how to do this, check out this blog post: https://sgillies.net/2017/10/19/mock-is-magic.html

2.3.5 Modify Rule

If you need to change the parameters of a Config rule in your working directory you can use the `modify` command. Any parameters you specify will overwrite existing values, any that you do not specify will not be changed.

```
$ rdk modify MyRule --runtime python2.7 --maximum-frequency TwentyFour_Hours --input-
↪parameters '{"desiredInstanceType":"t2.micro"}'
Running modify!
Modified Rule 'MyRule'.  Use the `deploy` command to push your changes to AWS.
```

Again, on Windows the input parameters would look like:

```
'{\"desiredInstanceType\":\"t2.micro\"}'
```

It is worth noting that until you actually call the `deploy` command your rule only exists in your working directory, none of the Rule commands discussed thus far actually makes changes to your account.

2.3.6 Deploy Rule

Once you have completed your compliance validation code and set your Rule's configuration, you can deploy the Rule to your account using the `deploy` command. This will zip up your code (and the other associated code files, if any) into a deployable package (or run a gradle build if you have selected the java8 runtime or run the lambda packaging step from the dotnet CLI if you have selected the dotnetcore1.0 runtime), copy that zip file to S3, and then launch or update a CloudFormation stack that defines your Config Rule, Lambda function, and the necessary permissions and IAM Roles for it to function. Since CloudFormation does not deeply inspect Lambda code objects in S3 to construct its changeset, the `deploy` command will also directly update the Lambda function for any subsequent deployments to make sure code changes are propagated correctly.

```
$ rdk deploy MyRule
Running deploy!
Zipping MyRule
Uploading MyRule
Creating CloudFormation Stack for MyRule
Waiting for CloudFormation stack operation to complete...
...
Waiting for CloudFormation stack operation to complete...
Config deploy complete.
```

The exact output will vary depending on Lambda runtime. You can use the –all flag to deploy all of the rules in your working directory.

2.3.7 View Logs For Deployed Rule

Once the Rule has been deployed to AWS you can get the CloudWatch logs associated with your lambda function using the `logs` command.

```
$ rdk logs MyRule -n 5
2017-11-15 22:59:33 - START RequestId: 96e7639a-ca15-11e7-95a2-b1521890638d Version:
↪$LATEST
2017-11-15 23:41:13 - REPORT RequestId: 68e0304f-ca1b-11e7-b735-81ebae95acda␣
↪Duration: 0.50 ms    Billed Duration: 100 ms     Memory Size: 256 MB
                      Max Memory Used: 36 MB
2017-11-15 23:41:13 - END RequestId: 68e0304f-ca1b-11e7-b735-81ebae95acda
2017-11-15 23:41:13 - Default RDK utility class does not yet support Scheduled␣
↪Notifications.
2017-11-15 23:41:13 - START RequestId: 68e0304f-ca1b-11e7-b735-81ebae95acda Version:
↪$LATEST
```

You can use the `-n` and `-f` command line flags just like the UNIX `tail` command to view a larger number of log events and to continuously poll for new events. The latter option can be useful in conjunction with manually initiating Config Evaluations for your deploy Config Rule to make sure it is behaving as expected.

2.4 Advanced Features

2.4.1 Cross-Account Deployments

Features have been added to the RDK to facilitate the cross-account deployment pattern that enterprise customers have standardized on for custom Config Rules. A cross-account architecture is one in which the Lambda functions are deployed to a single central "Compliance" account (which may be the same as a central "Security" account), and the Config Rules are deployed to any number of "Satellite" accounts that are used by other teams or departments. This gives the compliance team confidence that their Rule logic cannot be tampered with and makes it much easier for them to modify rule logic without having to go through a complex deployment process to potentially hundreds of AWS accounts. The cross-account pattern uses two advanced RDK features - functions-only deployments and the *create-rule-template* command.

Function-Only Deployment

By using the *-f* or *–functions-only* flag on the *deploy* command the RDK will deploy only the necessary Lambda Functions, Lambda Execution Role, and Lambda Permissions to the account specified by the execution credentials. It accomplishes this by batching up all of the Lambda function CloudFormation snippets for the selected Rule(s) into a single dynamically generated template and deploy that CloudFormation template. One consequence of this is that subsequent deployments that specify a different set of Rules for the same stack name will update that CloudFormation stack, and any Rules that were included in the first deployment but not in the second will be removed. You can use the *–stack-name* parameter to override the default CloudFormation stack name if you need to manage different subsets of your Lambda Functions independently. The intended usage is to deploy the functions for all of the Config rules in the Security/Compliance account, which can be done simply by using *rdk deploy -f –all* from your working directory.

'create-rule-template' command

This command generates a CloudFormation template that defines the AWS Config rules themselves, along with the Config Role, Config data bucket, Configuration Recorder, and Delivery channel necessary for the Config rules to work in a satellite account. You must specify the file name for the generated template using the *–output-file* or *o* command line flags. The generated template takes a single parameter of the AccountID of the central compliance account that contains the Lambda functions that will back your custom Config Rules. The generated template can be deployed in the desired satellite accounts through any of the means that you can deploy any other CloudFormation template, including the console, the CLI, as a CodePipeline task, or using StackSets. The *create-rule-template* command takes all of the standard arguments for selecting Rules to include in the generated template, including lists of individual Rule names, an *–all* flag, or using the RuleSets feature described below.

```
$ rdk create-rule-template -o remote-rule-template.json --all
Generating CloudFormation template!
CloudFormation template written to remote-rule-template.json
```

2.4.2 RuleSets

New as of version 0.3.11, it is possible to add RuleSet tags to rules that can be used to deploy and test groups of rules together. Rules can belong to multiple RuleSets, and RuleSet membership is stored only in the parameters.json metadata. The *deploy*, *create-rule-template*, and *test-local* commands are RuleSet-aware such that a RuleSet can be passed in as the target instead of *–all* or a specific named Rule.

A comma-delimited list of RuleSets can be added to a Rule when you create it (using the *–rulesets* flag), as part of a *modify* command, or using new *ruleset* subcommands to add or remove individual rules from a RuleSet.

Running *rdk rulesets list* will display a list of the RuleSets currently defined across all of the Rules in the working directory

```
rdk-dev $ rdk rulesets list
RuleSets:  AnotherRuleSet MyNewSet
```

Naming a specific RuleSet will list all of the Rules that are part of that RuleSet.

```
rdk-dev $ rdk rulesets list AnotherRuleSet
Rules in AnotherRuleSet :  RSTest
```

Rules can be added to or removed from RuleSets using the *add* and *remove* subcommands:

```
rdk-dev $ rdk rulesets add MyNewSet RSTest
RSTest added to RuleSet MyNewSet

rdk-dev $ rdk rulesets remove AnotherRuleSet RSTest
RSTest removed from RuleSet AnotherRuleSet
```

RuleSets are a convenient way to maintain a single repository of Config Rules that may need to have subsets of them deployed to different environments. For example your development environment may contain some of the Rules that you run in Production but not all of them; RuleSets gives you a way to identify and selectively deploy the appropriate Rules to each environment.

CHAPTER 3

Command Reference

The RDK has some options that can be used to override the default behavior (mostly relating to the identity and credentials used by the tool) that are common to all of the sub-commands.

```
usage: rdk [-h] [-p PROFILE] [-k ACCESS_KEY_ID] [-s SECRET_ACCESS_KEY]
           [-r REGION] [-v]
           <command> ...
```

3.1 Positional Arguments

<command> Possible choices: clean, create, create-rule-template, deploy, init, logs, modify, rulesets, sample-ci, test-local, undeploy

Command to run. Refer to the usage instructions for each command for more details

<command arguments> Run *rdk <command> –help* to see command-specific arguments.

3.2 Named Arguments

-p, --profile [optional] indicate which Profile to use.

-k, --access-key-id [optional] Access Key ID to use.

-s, --secret-access-key [optional] Secret Access Key to use.

-r, --region Select the region to run the command in.

-v, --version Display the version of this tool

3.3 Sub-Commands

3.3.1 Clean

The `clean` command is the inverse of the `init` command, and can be used to completely remove Config resources from an account, including the Configuration Recorder, Delivery Channel, S3 buckets, Roles, and Permissions. This is useful for testing account provisioning automation and for running automated tests in a clean environment.

```
usage: rdk clean [-h] [--force]
```

Named Arguments

--force [optional] Clean account without prompting for confirmation.

Default: False

3.3.2 Create

As of version 0.6, RDK supports Config remediation. Note that in order to use SSM documents for remediation you must supply all of the necessary document parameters. These can be found in the SSM document listing on the AWS console, but RDK will *not* validate at rule creation that you have all of the necessary parameters supplied.

Rules are stored in their own directory along with their metadata. This command is used to create the Rule and metadata.

```
usage: rdk create <rulename> --runtime <runtime> [ --resource-types <resource types>␣
↪| --maximum-frequency <max execution frequency> ] [optional configuration flags] [--
↪rulesets <RuleSet tags>]
```

Positional Arguments

<rulename> Rule name to create/modify

Named Arguments

-R, --runtime Possible choices: nodejs4.3, java8, python2.7, python3.6, python3.6-lib, python3.7, dotnetcore1.0, dotnetcore2.0

Runtime for lambda function

--source-identifier [optional] Used only for creating Managed Rules.

-r, --resource-types [optional] Resource types that will trigger event-based Rule evaluation

-m, --maximum-frequency Possible choices: One_Hour, Three_Hours, Six_Hours, Twelve_Hours, TwentyFour_Hours

[optional] Maximum execution frequency for scheduled Rules

-i, --input-parameters [optional] JSON for required Config parameters.

--optional-parameters [optional] JSON for optional Config parameters.

--tags [optional] JSON for tags to be applied to all CFN created resources.

-s, --rulesets [optional] comma-delimited list of RuleSet names to add this Rule to.

--remediation-action [optional] SSM document for remediation.

--remediation-action-version [optional] SSM document version for remediation action.

--auto-remediate [optional] Set the SSM remediation to trigger automatically.

Default: False

--auto-remediation-retry-attempts [optional] Number of times to retry automated remediation.

--auto-remediation-retry-time [optional] Duration of automated remediation retries.

--remediation-concurrent-execution-percent [optional] Concurrent execution rate of the SSM document for remediation.

--remediation-error-rate-percent [optional] Error rate that will mark the batch as "failed" for SSM remediation execution.

--remediation-resource-id-parameter [optional] Parameter that will be passed to SSM remediation document.

--remediation-parameters [optional] JSON-formatted string of additional parameters required by the SSM document.

3.3.3 Create-Rule-Template

Generates and saves to a file a single CloudFormation template that can be used to deploy the specified Rule(s) into any account. This feature has two primary uses:

- Multi-account Config setup in which the Lambda Functions for custom Rules are deployed into a centralized "security" or "compliance" account and the Config Rules themselves are deployed into "application" or "satellite" accounts.
- Combine many Config Rules into a single CloudFormation template for easier atomic deployment and management.

The generated CloudFormation template includes a Parameter for the AccountID that contains the Lambda functions that provide the compliance logic for the Rules, and also exposes all of the Config Rule input parameters as CloudFormation stack parameters.

By default the generated CloudFormation template will set up Config as per the settings used by the RDK `init` command, but those resources can be omitted using the `--rules-only` flag.

The `--config-role-arn` flag can be used for assigning existing config role to the created Configuration Recorder.

As of version 0.6, RDK supports Config remediation. Note that in order to use SSM documents for remediation you must supply all of the necessary document parameters. These can be found in the SSM document listing on the AWS console, but RDK will *not* validate at rule creation that you have all of the necessary parameters supplied.

```
usage: rdk create-rule-template [-h] [--all] [-s RULESETS] -o OUTPUT_FILE
                                [--config-role-arn CONFIG_ROLE_ARN]
                                [--rules-only]
                                [<rulename> [<rulename> ...]]
```

Positional Arguments

<rulename> Rule name(s) to include in template. A CloudFormation template will be created, but Rule(s) will not be pushed to AWS.

Named Arguments

--all, -a	All rules in the working directory will be included in the generated CloudFormation template. Default: False
-s, --rulesets	comma-delimited RuleSet names to be included in the generated template.
-o, --output-file	filename of generated CloudFormation template Default: "RDK-Config-Rules"
--config-role-arn	[optional] Assign existing iam role as config role. If omitted, "config-role" will be created.
--rules-only	[optional] Generate a CloudFormation Template that only includes the Config Rules and not the Bucket, Configuration Recorder, and Delivery Channel. Default: False

3.3.4 Deploy

This command will deploy the specified Rule(s) to the Account and Region determined by the credentials being used to execute the command, and the value of the AWS_DEFAULT_REGION environment variable, unless those credentials or region are overridden using the common flags.

Once deployed, RDK will _not_ explicitly start a Rule evaluation. Depending on the changes being made to your Config Rule setup AWS Config may re-evaluate the deployed Rules automatically, or you can run an evaluation using the AWS configservice CLI.

The `--lambda-role-arn` flag can be used for assigning existing iam role to all Lambda functions created for Custom Config Rules.

The `--functions-only` flag can be used as part of a multi-account deployment strategy to push _only_ the Lambda functions (and necessary Roles and Permssions) to the target account. This is intended to be used in conjunction with the `create-rule-template` command in order to separate the compliance logic from the evaluated accounts. For an example of how this looks in practice, check out the AWS Compliance-as-Code Engine.

Note: Behind the scenes the `--functions-only` flag generates a CloudFormation template and runs a "create" or "update" on the targeted AWS Account and Region. If subsequent calls to `deploy` with the `--functions-only` flag are made with the same stack name (either the default or otherwise) but with *different Config rules targeted*, any Rules deployed in previous `deploy``s but not included in the latest ``deploy` will be removed. After a functions-only `deploy` _only_ the Rules specifically targeted by that command (either through Rulesets or an explicit list supplied on the command line) will be deployed in the environment, all others will be removed.s

```
usage: rdk deploy [-h] [--all] [-s RULESETS] [-f]
                  [--lambda-role-arn LAMBDA_ROLE_ARN]
                  [--stack-name STACK_NAME]
                  [--execution-role-name EXECUTION_ROLE_NAME]
                  [--rdklib-layer-arn RDKLIB_LAYER_ARN]
                  [--lambda-layers LAMBDA_LAYERS]
                  [<rulename> [<rulename> ...]]
```

Positional Arguments

<rulename>	Rule name(s) to deploy. Rule(s) will be pushed to AWS.

Named Arguments

--all, -a All rules in the working directory will be deployed.

Default: False

-s, --rulesets comma-delimited list of RuleSet names

-f, --functions-only [optional] Only deploy Lambda functions. Useful for cross-account deployments.

Default: False

--lambda-role-arn [optional] Assign existing iam role to lambda functions. If omitted, "rdkLambdaRole" will be created.

--stack-name [optional] CloudFormation Stack name for use with –functions-only option. If omitted, "RDK-Config-Rule-Functions" will be used.

--execution-role-name [optional] IAM Role that the Lambda function(s) will assume in each target account.

--rdklib-layer-arn [optional] Lambda Layer ARN that contains the desired rdklib. Note that Lambda Layers are region-specific.

--lambda-layers [optional] Comma-separated list of Lambda Layer ARNs to deploy with your Lambda function(s).

3.3.5 Init

Sets up the AWS Config Service in an AWS Account. This includes:

- Config Configuration Recorder
- Config Delivery Channel
- IAM Role for Delivery Channel
- S3 Bucket for Configuration Snapshots
- S3 Bucket for Lambda Code

Additionally, `init` will make sure that the Configuration Recorder is on and functioning, that the Delivery Channel has the appropriate Role attached, and that the Delivery Channel Role has the proper permissions.

Note: Even without Config Rules running the Configuration Recorder is still capturing Configuration Item snapshots and storing them in S3, so running `init` will incur AWS charges!

Also Note: AWS Config is a regional service, so running `init` will only set up Config in the region currently specified in your AWS_DEFAULT_REGION environment variable or in the `--region` flag.

```
usage: rdk init [-h]
```

3.3.6 Logs

The `logs` command provides a shortcut to accessing the CloudWatch Logs output from the Lambda Functions that back your custom Config Rules. Logs are displayed in chronological order going back the number of log entries specified by the `--number` flag (default 3). It supports a `--follow` flag similar to the UNIX command `tail` so that you can choose to continually poll CloudWatch to deliver new log items as they are delivered by your Lambda function.

In addition to any output that your function emits via `print()` or `console.log()` commands, Lambda will also record log lines for the start and stop of each Lambda invocation, including the runtime and memory usage.

```
usage: rdk logs <rulename> [-n/--number NUMBER] [-f/--follow]
```

Positional Arguments

<rulename> Rule whose logs will be displayed

Named Arguments

-f, --follow [optional] Continuously poll Lambda logs and write to stdout.

Default: False

-n, --number [optional] Number of previous logged events to display.

Default: 3

3.3.7 Modify

Used to modify the local metadata for Config Rules created by the RDK. This command takes the same arguments as the `create` command (all of them optional), and overwrites the Rule metadata for any flag specified. Changes made using `modify` are not automatically pushed out to your AWS Account, and must be deployed as usual using the `deploy` command.

```
usage: rdk modify <rulename> [--runtime <runtime>] [--resource-types <resource types>
↪] [--maximum-frequency <max execution frequency>] [--input-parameters <parameter␣
↪JSON>] [--tags <tags JSON>] [--rulesets <RuleSet tags>]
```

Positional Arguments

<rulename> Rule name to create/modify

Named Arguments

-R, --runtime Possible choices: nodejs4.3, java8, python2.7, python3.6, python3.6-lib, python3.7, dotnetcore1.0, dotnetcore2.0

Runtime for lambda function

--source-identifier [optional] Used only for creating Managed Rules.

-r, --resource-types [optional] Resource types that will trigger event-based Rule evaluation

-m, --maximum-frequency Possible choices: One_Hour, Three_Hours, Six_Hours, Twelve_Hours, TwentyFour_Hours

[optional] Maximum execution frequency for scheduled Rules

-i, --input-parameters [optional] JSON for required Config parameters.

--optional-parameters [optional] JSON for optional Config parameters.

--tags [optional] JSON for tags to be applied to all CFN created resources.

-s, --rulesets [optional] comma-delimited list of RuleSet names to add this Rule to.

--remediation-action [optional] SSM document for remediation.

--remediation-action-version [optional] SSM document version for remediation action.

--auto-remediate [optional] Set the SSM remediation to trigger automatically.

Default: False

--auto-remediation-retry-attempts [optional] Number of times to retry automated remediation.

--auto-remediation-retry-time [optional] Duration of automated remediation retries.

--remediation-concurrent-execution-percent [optional] Concurrent execution rate of the SSM document for remediation.

--remediation-error-rate-percent [optional] Error rate that will mark the batch as "failed" for SSM remediation execution.

--remediation-resource-id-parameter [optional] Parameter that will be passed to SSM remediation document.

--remediation-parameters [optional] JSON-formatted string of additional parameters required by the SSM document.

3.3.8 Rulesets

Rulesets provide a mechanism to tag individual Config Rules into groups that can be acted on as a unit. Ruleset tags are single keywords, and the commands `deploy`, `create-rule-template`, and `undeploy` can all expand Ruleset parameters and operate on the resulting list of Rules.

The most common use-case for Rulesets is to define standardized Account metadata or data classifications, and then tag individual Rules to all of the appropriate metadata tags or classification levels.

Example: If you have Account classifications of "Public", "Private", and "Restricted" you can tag all of your Rules as "Restricted", and a subset of them that deal with private network security as "Private". Then when you need to deploy controls to a new "Private" account you can simply use `rdk create-rule-template --rulesets Private` to generate a CloudFormation template that includes all of the Rules necessary for your "Private" classification, but omit the Rules that are only necessary for "Restricted" accounts. Additionally, as your compliance requirements change and you add Config Rules you can tag them as appropriate, re-generate your CloudFormation templates, and re-deploy to make sure your Accounts are all up-to-date.

You may also choose to classify accounts using binary attributes ("Prod" vs. "Non-Prod" or "PCI" vs. "Non-PCI"), and then generate account-specific CloudFormation templates using the Account metadata to ensure that the appropriate controls are deployed.

```
usage: rdk rulesets [list | [ [ add | remove ] <ruleset> <rulename> ]
```

Positional Arguments

subcommand One of list, add, or remove

ruleset Name of RuleSet

rulename Name of Rule to be added or removed

3.3.9 Sample-CI

This utility command outputs a sample Configuration Item for the specified resource type. This can be useful when writing new custom Config Rules to help developers know what the CI structure and plausible values for the resource type are.

Note that you can construct Config Evaluations for any resource type that is supported by CloudFormation, however you can not create change-triggered Config Rules for resource types not explicitly supported by Config, and some of the console functionality in AWS Config may be limited.

CFN-supported resources Config-supported resources

```
usage: rdk sample-ci [-h] <resource type>
```

Positional Arguments

<resource type> Possible choices: AWS::ACM::Certificate, AWS::ApiGateway::RestApi, AWS::ApiGateway::Stage, AWS::ApiGatewayV2::Api, AWS::ApiGatewayV2::Stage, AWS::AutoScaling::AutoScalingGroup, AWS::AutoScaling::LaunchConfiguration, AWS::AutoScaling::ScalingPolicy, AWS::AutoScaling::ScheduledAction, AWS::CloudFormation::Stack, AWS::CloudFront::Distribution, AWS::CloudFront::StreamingDistribution, AWS::CloudTrail::Trail, AWS::CloudWatch::Alarm, AWS::CodeBuild::Project, AWS::CodePipeline::Pipeline, AWS::DynamoDB::Table, AWS::EC2::CustomerGateway, AWS::EC2::EIP, AWS::EC2::EgressOnlyInternetGateway, AWS::EC2::FlowLog, AWS::EC2::Host, AWS::EC2::Instance, AWS::EC2::InternetGateway, AWS::EC2::NatGateway, AWS::EC2::NetworkAcl, AWS::EC2::NetworkInterface, AWS::EC2::RouteTable, AWS::EC2::SecurityGroup, AWS::EC2::Subnet, AWS::EC2::VPC, AWS::EC2::VPCEndpoint, AWS::EC2::VPCEndpointService, AWS::EC2::VPCPeeringConnection, AWS::EC2::VPNConnection, AWS::EC2::VPNGateway, AWS::EC2::Volume, AWS::ElasticBeanstalk::Application, AWS::ElasticBeanstalk::ApplicationVersion, AWS::ElasticBeanstalk::Environment, AWS::ElasticLoadBalancing::LoadBalancer, AWS::ElasticLoadBalancingV2::LoadBalancer, AWS::IAM::Group, AWS::IAM::Policy, AWS::IAM::Role, AWS::IAM::User, AWS::Lambda::Function, AWS::QLDB::Ledger, AWS::RDS::DBCluster, AWS::RDS::DBClusterSnapshot, AWS::RDS::DBInstance, AWS::RDS::DBSecurityGroup, AWS::RDS::DBSnapshot, AWS::RDS::DBSubnetGroup, AWS::RDS::EventSubscription, AWS::Redshift::Cluster, AWS::Redshift::ClusterParameterGroup, AWS::Redshift::ClusterSecurityGroup, AWS::Redshift::ClusterSnapshot, AWS::Redshift::ClusterSubnetGroup, AWS::Redshift::EventSubscription, AWS::S3::AccountPublicAccessBlock, AWS::S3::Bucket, AWS::SSM::AssociationCompliance, AWS::SSM::ManagedInstanceInventory, AWS::SSM::PatchCompliance, AWS::ServiceCatalog::CloudFormationProduct, AWS::ServiceCatalog::CloudFormationProvisionedProduct, AWS::ServiceCatalog::Portfolio, AWS::Shield::Protection, AWS::ShieldRegional::Protection, AWS::WAF::RateBasedRule, AWS::WAF::Rule, AWS::WAF::RuleGroup, AWS::WAF::WebACL, AWS::WAFRegional::RateBasedRule, AWS::WAFRegional::Rule, AWS::WAFRegional::RuleGroup, AWS::WAFRegional::WebACL, AWS::XRay::EncryptionConfig

Resource name (e.g. "AWS::EC2::Instance") to display a sample CI JSON document for.

3.3.10 Test-Local

Shorthand command for running the unit tests defined for Config Rules that use a Python runtime. When a Python 2.7 or 3.6+ Rule is created using the `create` command a unit test template is created in the Rule directory. This test boilerplate includes minimal tests, as well as a framework for using the `mock` library for stubbing out Boto3 calls. This allows more sophisticated test cases to be written for Periodic rules that need to make API calls to gather information about the environment.

```
usage: rdk test-local [-h] [--all] [--test-ci-json TEST_CI_JSON]
                      [--test-ci-types TEST_CI_TYPES] [--verbose]
                      [-s RULESETS]
                      [<rulename>[,<rulename>,...]
                      [<rulename>[,<rulename>,...] ...]]
```

Positional Arguments

<rulename>[,<rulename>,...] Rule name(s) to test

Named Arguments

--all, -a	Test will be run against all rules in the working directory.
	Default: False
--test-ci-json, -j	[optional] JSON for test CI for testing.
--test-ci-types, -t	[optional] CI type to use for testing.
--verbose, -v	[optional] Enable full log output
	Default: False
-s, --rulesets	[p[tional] comma-delimited list of RuleSet names

3.3.11 Undeploy

The inverse of `deploy`, this command is used to remove a Config Rule and its Lambda Function from the targeted account.

This is intended to be used primarily for clean-up for testing deployment automation (perhaps from a CI/CD pipeline) to ensure that it works from an empty account, or to clean up a test account during development. See also the clean command if you want to more thoroughly scrub Config from your account.

```
usage: rdk undeploy [-h] [--all] [-s RULESETS] [-f]
                    [--lambda-role-arn LAMBDA_ROLE_ARN]
                    [--stack-name STACK_NAME]
                    [--execution-role-name EXECUTION_ROLE_NAME]
                    [--rdklib-layer-arn RDKLIB_LAYER_ARN]
                    [--lambda-layers LAMBDA_LAYERS] [--force]
                    [<rulename> [<rulename> ...]]
```

Positional Arguments

<rulename> Rule name(s) to deploy. Rule(s) will be pushed to AWS.

Named Arguments

--all, -a All rules in the working directory will be deployed.

Default: False

-s, --rulesets comma-delimited list of RuleSet names

-f, --functions-only [optional] Only deploy Lambda functions. Useful for cross-account deployments.

Default: False

--lambda-role-arn [optional] Assign existing iam role to lambda functions. If omitted, "rdkLambdaRole" will be created.

--stack-name [optional] CloudFormation Stack name for use with –functions-only option. If omitted, "RDK-Config-Rule-Functions" will be used.

--execution-role-name [optional] IAM Role that the Lambda function(s) will assume in each target account.

--rdklib-layer-arn [optional] Lambda Layer ARN that contains the desired rdklib. Note that Lambda Layers are region-specific.

--lambda-layers [optional] Comma-separated list of Lambda Layer ARNs to deploy with your Lambda function(s).

--force [optional] Remove selected Rules from account without prompting for confirmation.

Default: False

CHAPTER 4

Indices and tables

- genindex
- modindex
- search