IAM Floyd

Release 0.76.0

Daniel Schroeder

Oct 06, 2020

CONTENTS

Attention: This is an early version of the package. The API might change when new features are implemented. Therefore make sure you use an exact version in your `package.json/requirements.txt` before it reaches 1.0.0.

There are two different package variants available:

iam-floyd:

Can be used in AWS SDK, Boto 3 or for whatever you need an IAM policy statement for:

cdk-iam-floyd:

Integrates into AWS CDK and extends iam.PolicyStatement:

Find them all on libraries.io.

Java packages currently are only available on GitHub.

CHAPTER ONE

GETTING STARTED

Attention: This is an early version of the package. The API might change when new features are implemented. Therefore make sure you use an exact version in your `package.json`/`requirements.txt` before it reaches 1.0.0.

Depending on your scenario, you need to either install/import `iam-floyd` or `cdk-iam-floyd`:

JavaScript

```
# for use without AWS CDK use the iam-floyd package
npm install iam-floyd

# for use with CDK use the cdk-iam-floyd package
npm install cdk-iam-floyd
```

Python

```
# for use without AWS CDK use the iam-floyd package
pip install iam-floyd

# for use with CDK use the cdk-iam-floyd package
pip install cdk-iam-floyd
```

TypeScript

```
// for use without AWS CDK use the iam-floyd package
import * as statement from 'iam-floyd';

// for use with CDK use the cdk-iam-floyd package
import * as statement from 'cdk-iam-floyd';
```

JavaScript

```
// for use without AWS CDK use the iam-floyd package
var statement = require('iam-floyd');

// for use with CDK use the cdk-iam-floyd package
var statement = require('cdk-iam-floyd');
```

Python

```
# for use without AWS CDK use the iam-floyd package
import iam_floyd as statement
```

(continues on next page)

(continued from previous page)

```
# for use with CDK use the cdk-iam-floyd package
import cdk_iam_floyd as statement
```

Both packages contain a statement provider for each AWS service, e.g. `Ec2`. A statement provider is a class with methods for each and every available action, resource type and condition. Calling such method will add the action/resource/condition to the statement:

TypeScript

```
new statement.Ec2().toStartInstances();
```

JavaScript

```
new statement.Ec2().toStartInstances();
```

Python

```
statement.Ec2().to_start_instances()
```

Every method returns the statement provider, so you can chain method calls:

TypeScript

```
new statement.Ec2() //
  .toStartInstances()
  .toStopInstances();
```

JavaScript

```
new statement.Ec2() //
  .toStartInstances()
  .toStopInstances();
```

Python

```
statement.Ec2() \
    .to_start_instances() \
    .to_stop_instances()
```

The default effect of any statement is `Allow`. To add some linguistic sugar you can explicitly call the `allow()` method:

TypeScript

```
new statement.Ec2() //
  .allow()
  .toStartInstances()
  .toStopInstances();
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .toStartInstances()
  .toStopInstances();
```

Python

```
statement.Ec2() \
    .allow() \
    .to_start_instances() \
    .to_stop_instances()
```

Or `deny()`:

TypeScript

```
new statement.Ec2() //
  .deny()
  .toStartInstances()
  .toStopInstances();
```

JavaScript

```
new statement.Ec2() //
  .deny()
  .toStartInstances()
  .toStopInstances();
```

Python

```
statement.Ec2() \
    .deny() \
    .to_start_instances() \
    .to_stop_instances()
```

You can work with access levels. For every access level there are distinct methods available to add all related actions to the statement:

TypeScript

- `allListActions()`
- `allReadActions()`
- `allWriteActions()`
- `allPermissionManagementActions()`
- `allTaggingActions()`

JavaScript

- `allListActions()`
- `allReadActions()`
- `allWriteActions()`
- `allPermissionManagementActions()`
- `allTaggingActions()`

Python

- `all_list_actions()`
- `all_read_actions()`
- `all_write_actions()`
- `all_permission_management_actions()`

- `all_tagging_actions()`

TypeScript

```
new statement.Ec2() //
  .deny()
  .allPermissionManagementActions();

new statement.Ec2() //
  .allow()
  .allListActions()
  .allReadActions();
```

JavaScript

```
new statement.Ec2() //
  .deny()
  .allPermissionManagementActions();

new statement.Ec2() //
  .allow()
  .allListActions()
  .allReadActions();
```

Python

```
statement.Ec2() \
    .deny() \
    .all_permission_management_actions()

statement.Ec2() \
    .allow() \
    .all_list_actions() \
    .all_read_actions()
```

To add actions based on regular expressions, use the method `allMatchingActions()`.

Important: No matter in which language you use the package, the regular expressions need to be in Perl/JavaScript literal style and need to be passed as strings!

TypeScript

```
new statement.Ec2() //
  .deny()
  .allMatchingActions('/vpn/i');
```

JavaScript

```
new statement.Ec2() //
  .deny()
  .allMatchingActions('/vpn/i');
```

Python

```
statement.Ec2() \
    .deny() \
    .all_matching_actions('/vpn/i')
```

To add all actions (e.g. `ec2:*`), call the `allActions()` method:

TypeScript

```
new statement.Ec2() //
  .allow()
  .allActions();
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .allActions();
```

Python

```
statement.Ec2() \
    .allow() \
    .all_actions()
```

For every available condition key, there are `if*()` methods available.

TypeScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .ifEncrypted()
  .ifInstanceType(['t3.micro', 't3.nano'])
  .ifAssociatePublicIpAddress(false)
  .ifAwsRequestTag('Owner', 'John');
```

JavaScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .ifEncrypted()
  .ifInstanceType(['t3.micro', 't3.nano'])
  .ifAssociatePublicIpAddress(false)
  .ifAwsRequestTag('Owner', 'John');
```

Python

```
statement.Ec2() \
    .allow() \
    .to_start_instances() \
    .if_encrypted() \
    .if_instance_type(['t3.micro', 't3.nano']) \
    .if_associate_public_ip_address(False) \
    .if_aws_request_tag('Owner', 'John')
```

To add a condition not covered by the available methods, you can define just any condition yourself via `if()`:

TypeScript

```
new statement.Ec2()
  .allow()
```

(continues on next page)

(continued from previous page)

```
  .toStartInstances()
  .if('ec2:missingCondition', 'some-value');
```

JavaScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .if('ec2:missingCondition', 'some-value');
```

Python

```
statement.Ec2() \
    .allow() \
    .to_start_instances(). \
    if_('ec2:missingCondition', 'some-value')
```

The default operator for conditions of type String is StringLike.

Most of the `if*()` methods allow an optional operator as last argument:

TypeScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .ifAwsRequestTag('TagWithSpecialChars', '*John*', 'StringEquals');
```

JavaScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .ifAwsRequestTag('TagWithSpecialChars', '*John*', 'StringEquals');
```

Python

```
statement.Ec2() \
    .allow() \
    .to_start_instances() \
    .if_aws_request_tag('TagWithSpecialChars', '*John*', 'StringEquals')
```

Statements without principals, by default, apply to all resources. To limit to specific resources, add them via `on*()`. For every resource type an `on*()` method exists:

TypeScript

```
new statement.S3()
  .allow()
  .allActions()
  .onBucket('some-bucket')
  .onObject('some-bucket', 'some/path/*');
```

JavaScript

```
new statement.S3()
  .allow()
```

(continues on next page)

(continued from previous page)

```
  .allActions()
  .onBucket('some-bucket')
  .onObject('some-bucket', 'some/path/*');
```

Python

```
statement.S3() \
    .allow() \
    .all_actions() \
    .on_bucket('some-bucket') \
    .on_object('some-bucket', 'some/path/*')
```

If instead you have an ARN ready, use the `on()` method:

TypeScript

```
new statement.S3() //
  .allow()
  .allActions()
  .on(
    'arn:aws:s3:::some-bucket', //
    'arn:aws:s3:::another-bucket'
  );
```

JavaScript

```
new statement.S3() //
  .allow()
  .allActions()
  .on(
    'arn:aws:s3:::some-bucket', //
    'arn:aws:s3:::another-bucket'
  );
```

Python

```
statement.S3() \
    .allow() \
    .all_actions() \
    .on('arn:aws:s3:::some-bucket',
        'arn:aws:s3:::another-bucket')
```

To invert the policy you can use `notActions()`, `notResources()` and `notPrincipals()`:

TypeScript

```
new statement.S3()
  .allow()
  .notActions()
  .toDeleteBucket()
  .onBucket('some-bucket');
```

JavaScript

```
new statement.S3()
  .allow()
  .notActions()
```

(continues on next page)

(continued from previous page)

```
  .toDeleteBucket()
  .onBucket('some-bucket');
```

Python

```
statement.S3() \
    .allow() \
    .not_actions() \
    .to_delete_bucket() \
    .on_bucket('some-bucket')
```

TypeScript

```
new statement.S3()
  .allow()
  .notResources()
  .toDeleteBucket()
  .onBucket('some-bucket');
```

JavaScript

```
new statement.S3()
  .allow()
  .notResources()
  .toDeleteBucket()
  .onBucket('some-bucket');
```

Python

```
statement.S3() \
    .allow() \
    .not_resources() \
    .to_delete_bucket() \
    .on_bucket('some-bucket')
```

TypeScript

```
new statement.Sts()
  .deny()
  .notPrincipals()
  .toAssumeRole()
  .forUser('1234567890', 'Bob');
```

JavaScript

```
new statement.Sts()
  .deny()
  .notPrincipals()
  .toAssumeRole()
  .forUser('1234567890', 'Bob');
```

Python

```
statement.Sts() \
    .deny() \
    .not_principals() \
```

(continues on next page)

(continued from previous page)

```
    .to_assume_role() \
    .for_user('1234567890', 'Bob')
```

CHAPTER

TWO

VOCABULARY

Attention: This is an early version of the package. The API might change when new features are implemented. Therefore make sure you use an exact version in your `package.json`/`requirements.txt` before it reaches 1.0.0.

IAM Floyd provides a fluid interface and enables you to define policy statements in a human readable and easy to understand phrase.

2.1 allow | deny (Effect)

The methods `allow()` and `deny()` control the Effect of the statement.

The default effect of any statement is `Allow`, so it's not mandatory to add either of these methods to the method chain. Though it is recommended to keep the statement readable. improve , so the statement is readable:

TypeScript

```
new statement.Ec2() //
  .allow()
  .toStartInstances();

new statement.Ec2() //
  .deny()
  .toStopInstances();
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .toStartInstances();

new statement.Ec2() //
  .deny()
  .toStopInstances();
```

Python

```
statement.Ec2() \
    .allow() \
    .to_start_instances()
```

(continues on next page)

(continued from previous page)

```
statement.Ec2() \
    .deny() \
    .to_stop_instances()
```

2.2 to (Action)

Every available IAM action is represented by a distinct method. These methods start with `to`. You allow/deny **to** *do something*

TypeScript

```
new statement.Ec2() //
  .allow()
  .toStartInstances()
  .toStopInstances();
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .toStartInstances()
  .toStopInstances();
```

Python

```
statement.Ec2() \
    .allow() \
    .to_start_instances() \
    .to_stop_instances()
```

In case of missing actions, you can just add any action key yourself via `to()`:

TypeScript

```
new statement.Ec2() //
  .allow()
  .to('missingAction');
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .to('missingAction');
```

Python

```
statement.Ec2() \
    .allow() \
    .to('missingAction')
```

2.3 all (Action)

While methods starting with `to` add a single action to a statement, methods starting with `all` add multiple actions.

2.3.1 allActions

This method adds all actions of the related service to the statement, e.g. *ec2:**

TypeScript

```
new statement.Ec2() //
  .allow()
  .allActions();
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .allActions();
```

Python

```
statement.Ec2() \
    .allow() \
    .all_actions()
```

2.3.2 allMatchingActions

Adds all actions matching regular expressions to the statement.

Attention: The list of actions is compiled at run time. The generated statement object contains an exact list of actions that matched when you build it. If AWS later adds/removes actions that would match the regular expression, you need to re-generate the statements.

The regular expressions need to be in Perl/JavaScript literal style and need to be passed as strings:

TypeScript

```
new statement.Ec2() //
  .deny()
  .allMatchingActions('/vpn/i');
```

JavaScript

```
new statement.Ec2() //
  .deny()
  .allMatchingActions('/vpn/i');
```

Python

```
statement.Ec2() \
    .deny() \
    .all_matching_actions('/vpn/i')
```

2.3.3 Access levels

To add all actions of a certain access level to the statement use the below methods.

> **Attention:** The list of actions is compiled at run time. The generated statement object contains an exact list of actions that matched when you build it. If AWS later adds/removes actions or changes the level, you need to re-generate the statements.

allListActions

Adds all actions with access level **list** to the statement.

TypeScript

```
new statement.Ec2() //
  .allow()
  .allListActions();
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .allListActions();
```

Python

```
statement.Ec2() \
     .allow() \
     .all_list_actions()
```

allReadActions

Adds all actions with access level **read** to the statement.

TypeScript

```
new statement.Ec2() //
  .allow()
  .allReadActions();
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .allReadActions();
```

Python

```
statement.Ec2() \
     .allow() \
     .all_read_actions()
```

allWriteActions

Adds all actions with access level **write** to the statement.

TypeScript

```
new statement.Ec2() //
  .allow()
  .allWriteActions();
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .allWriteActions();
```

Python

```
statement.Ec2() \
    .allow() \
    .all_write_actions()
```

allPermissionManagementActions

Adds all actions with access level **permission management** to the statement.

TypeScript

```
new statement.Ec2() //
  .allow()
  .allPermissionManagementActions();
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .allPermissionManagementActions();
```

Python

```
statement.Ec2() \
    .allow() \
    .all_permission_management_actions()
```

allTaggingActions

Adds all actions with access level **tagging** to the statement.

TypeScript

```
new statement.Ec2() //
  .allow()
  .allTaggingActions();
```

JavaScript

```
new statement.Ec2() //
  .allow()
  .allTaggingActions();
```

Python

```
statement.Ec2() \
    .allow() \
    .all_tagging_actions()
```

2.4 if (Condition)

Every available IAM condition key is represented by a distinct method. These methods start with `if`. You allow/deny something **if** a condition is met.

Every statement provider (e.g. `Ec2`) brings its unique conditions. Global condition context keys start with `ifAws`.

Note: Multiple conditions on a statement all have to be true.

When you have multiple values on a single condition, one of them has to be true.

Other than that, IAM has no concept of `OR`. You need to define multiple statements for each `OR` branch.

TypeScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .ifEncrypted()
  .ifInstanceType(['t3.micro', 't3.nano'])
  .ifAssociatePublicIpAddress(false)
  .ifAwsRequestTag('Owner', 'John');
```

JavaScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .ifEncrypted()
  .ifInstanceType(['t3.micro', 't3.nano'])
  .ifAssociatePublicIpAddress(false)
  .ifAwsRequestTag('Owner', 'John');
```

Python

```
statement.Ec2() \
    .allow() \
    .to_start_instances() \
    .if_encrypted() \
    .if_instance_type(['t3.micro', 't3.nano']) \
    .if_associate_public_ip_address(False) \
    .if_aws_request_tag('Owner', 'John')
```

Every `if` method has a default operator. For instance, conditions which operate on strings usually have `StringLike` as default. Most methods allow you to pass an operator as last argument.

TypeScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .ifAwsRequestTag('TagWithSpecialChars', '*John*', 'StringEquals');
```

JavaScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .ifAwsRequestTag('TagWithSpecialChars', '*John*', 'StringEquals');
```

Python

```
statement.Ec2() \
    .allow() \
    .to_start_instances() \
    .if_aws_request_tag('TagWithSpecialChars', '*John*', 'StringEquals')
```

In case of missing conditions, you can define just any condition yourself via `if()`:

TypeScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .if('ec2:missingCondition', 'some-value');
```

JavaScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .if('ec2:missingCondition', 'some-value');
```

Python

```
statement.Ec2() \
    .allow() \
    .to_start_instances(). \
    if_('ec2:missingCondition', 'some-value')
```

2.4.1 Operators

Condition operators can just be passed as strings. Or you can use the class `statement.Operator()`:

TypeScript

```
new statement.Dynamodb()
  .allow()
  .toGetItem()
  .onTable('Thread')
  .ifAttributes(
    ['ID', 'Message', 'Tags'],
```

(continues on next page)

(continued from previous page)

```
    new statement.Operator().stringEquals().forAllValues()
  );
```

JavaScript

```
new statement.Dynamodb()
  .allow()
  .toGetItem()
  .onTable('Thread')
  .ifAttributes(
    ['ID', 'Message', 'Tags'],
    new statement.Operator().stringEquals().forAllValues()
  );
```

Python

```
statement.Dynamodb() \
    .allow() \
    .to_get_item() \
    .on_table('Thread') \
    .if_attributes(['ID', 'Message', 'Tags'],
                   statement.Operator().string_equals().for_all_values())
```

TypeScript

```
new statement.Dynamodb()
  .deny()
  .toPutItem()
  .onTable('Thread')
  .ifAttributes(
    ['ID', 'PostDateTime'],
    new statement.Operator().stringEquals().forAnyValue()
  );
```

JavaScript

```
new statement.Dynamodb()
  .deny()
  .toPutItem()
  .onTable('Thread')
  .ifAttributes(
    ['ID', 'PostDateTime'],
    new statement.Operator().stringEquals().forAnyValue()
  );
```

Python

```
statement.Dynamodb() \
    .deny() \
    .to_put_item() \
    .on_table('Thread') \
    .if_attributes(['ID', 'PostDateTime'],
                   statement.Operator().string_equals().for_any_value())
```

TypeScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .ifAwsRequestTag(
    'Environment',
    ['Production', 'Staging', 'Dev'],
    new statement.Operator().stringEquals().ifExists()
  );
```

JavaScript

```
new statement.Ec2()
  .allow()
  .toStartInstances()
  .ifAwsRequestTag(
    'Environment',
    ['Production', 'Staging', 'Dev'],
    new statement.Operator().stringEquals().ifExists()
  );
```

Python

```
statement.Ec2() \
    .allow() \
    .to_start_instances() \
    .if_aws_request_tag('Environment',
                        ['Production', 'Staging', 'Dev'],
                        statement.Operator().string_equals().if_exists())
```

2.5 on (Resource)

Every available IAM resources key is represented by a distinct method. These methods start with `on`. You allow/deny something **on** a specific resource (or pattern).

TypeScript

```
new statement.S3()
  .allow()
  .allActions()
  .onBucket('some-bucket')
  .onObject('some-bucket', 'some/path/*');
```

JavaScript

```
new statement.S3()
  .allow()
  .allActions()
  .onBucket('some-bucket')
  .onObject('some-bucket', 'some/path/*');
```

Python

```
statement.S3() \
    .allow() \
    .all_actions() \
```

(continues on next page)

(continued from previous page)

```
    .on_bucket('some-bucket') \
    .on_object('some-bucket', 'some/path/*')
```

In case of missing resources or if you already have an ARN ready, use the `on()` method:

TypeScript

```
new statement.S3() //
  .allow()
  .allActions()
  .on(
    'arn:aws:s3:::some-bucket', //
    'arn:aws:s3:::another-bucket'
  );
```

JavaScript

```
new statement.S3() //
  .allow()
  .allActions()
  .on(
    'arn:aws:s3:::some-bucket', //
    'arn:aws:s3:::another-bucket'
  );
```

Python

```
statement.S3() \
    .allow() \
    .all_actions() \
    .on('arn:aws:s3:::some-bucket',
        'arn:aws:s3:::another-bucket')
```

If no resources are applied to the statement without principals, it defaults to all resources (`*`).

2.6 for (Principal)

Note: If you use the CDK variant of the package, don't attempt to create an assume policy with this package. Assume policies have to be of type `IPrincipal` and can easily be created with the iam package.

Every possible principal is represented by a distinct method. These methods start with `for`. You allow/deny something **for** a specific principal.

TypeScript

```
new statement.Sts() //
  .allow()
  .toAssumeRole()
  .forAccount('1234567890');

new statement.Sts()
  .allow()
  .toAssumeRoleWithSAML()
```

(continues on next page)

(continued from previous page)

```
  .forService('lambda.amazonaws.com');

new statement.Sts() //
  .allow()
  .toAssumeRole()
  .forUser('1234567890', 'Bob');

new statement.Sts() //
  .allow()
  .toAssumeRole()
  .forRole('1234567890', 'role-name');

new statement.Sts() //
  .allow()
  .toAssumeRoleWithSAML()
  .forFederatedCognito();

new statement.Sts() //
  .allow()
  .toAssumeRoleWithSAML()
  .forFederatedAmazon();

new statement.Sts() //
  .allow()
  .toAssumeRoleWithSAML()
  .forFederatedGoogle();

new statement.Sts() //
  .allow()
  .toAssumeRoleWithSAML()
  .forFederatedFacebook();

new statement.Sts()
  .allow()
  .toAssumeRoleWithSAML()
  .forSaml('1234567890', 'saml-provider');

new statement.Sts() //
  .allow()
  .toAssumeRole()
  .forPublic();

new statement.Sts()
  .allow()
  .toAssumeRole()
  .forAssumedRoleSession('123456789', 'role-name', 'session-name');

new statement.Sts() //
  .allow()
  .toAssumeRole()
  .forCanonicalUser('userID');

new statement.Sts() //
  .allow()
  .toAssumeRole()
  .for('arn:foo:bar');
```

JavaScript

```
new statement.Sts() //
  .allow()
  .toAssumeRole()
  .forAccount('1234567890');

new statement.Sts()
  .allow()
  .toAssumeRoleWithSAML()
  .forService('lambda.amazonaws.com');

new statement.Sts() //
  .allow()
  .toAssumeRole()
  .forUser('1234567890', 'Bob');

new statement.Sts() //
  .allow()
  .toAssumeRole()
  .forRole('1234567890', 'role-name');

new statement.Sts() //
  .allow()
  .toAssumeRoleWithSAML()
  .forFederatedCognito();

new statement.Sts() //
  .allow()
  .toAssumeRoleWithSAML()
  .forFederatedAmazon();

new statement.Sts() //
  .allow()
  .toAssumeRoleWithSAML()
  .forFederatedGoogle();

new statement.Sts() //
  .allow()
  .toAssumeRoleWithSAML()
  .forFederatedFacebook();

new statement.Sts()
  .allow()
  .toAssumeRoleWithSAML()
  .forSaml('1234567890', 'saml-provider');

new statement.Sts() //
  .allow()
  .toAssumeRole()
  .forPublic();

new statement.Sts()
  .allow()
  .toAssumeRole()
  .forAssumedRoleSession('123456789', 'role-name', 'session-name');

new statement.Sts() //
```

(continues on next page)

(continued from previous page)

```
  .allow()
  .toAssumeRole()
  .forCanonicalUser('userID');

new statement.Sts() //
  .allow()
  .toAssumeRole()
  .for('arn:foo:bar');
```

Python

```
statement.Sts() \
    .allow() \
    .to_assume_role() \
    .for_account('1234567890')

statement.Sts() \
    .allow() \
    .to_assume_role_with_saml() \
    .for_service('lambda.amazonaws.com')

statement.Sts() \
    .allow() \
    .to_assume_role() \
    .for_user('1234567890', 'Bob')

statement.Sts() \
    .allow() \
    .to_assume_role() \
    .for_role('1234567890', 'role-name')

statement.Sts() \
    .allow() \
    .to_assume_role_with_saml() \
    .for_federated_cognito()

statement.Sts() \
    .allow() \
    .to_assume_role_with_saml() \
    .for_federated_amazon()

statement.Sts() \
    .allow() \
    .to_assume_role_with_saml() \
    .for_federated_google()

statement.Sts() \
    .allow() \
    .to_assume_role_with_saml() \
    .for_federated_facebook()

statement.Sts() \
    .allow() \
    .to_assume_role_with_saml() \
    .for_saml('1234567890', 'saml-provider')

statement.Sts() \
```

(continues on next page)

(continued from previous page)

```
    .allow() \
    .to_assume_role() \
    .for_public()

statement.Sts() \
    .allow() \
    .to_assume_role() \
    .for_assumed_role_session('123456789', 'role-name', 'session-name')

statement.Sts() \
    .allow() \
    .to_assume_role() \
    .for_canonical_user('userID')

statement.Sts() \
    .allow() \
    .to_assume_role() \
    .for_('arn:foo:bar')
```

The CDK variant of the package has an additional method `forCdkPrincipal`, which takes any number of iam.IPrincipal objects:

TypeScript

```
new statement.Sts()
  .allow()
  .toAssumeRole()
  .forCdkPrincipal(
    new iam.ServicePrincipal('sns.amazonaws.com'),
    new iam.ServicePrincipal('lambda.amazonaws.com')
  );
```

JavaScript

```
new statement.Sts()
  .allow()
  .toAssumeRole()
  .forCdkPrincipal(
    new iam.ServicePrincipal('sns.amazonaws.com'),
    new iam.ServicePrincipal('lambda.amazonaws.com')
  );
```

Python

```
statement.Sts() \
    .allow() \
    .to_assume_role() \
    .for_cdk_principal(iam.ServicePrincipal('sns.amazonaws.com'),
                       iam.ServicePrincipal('lambda.amazonaws.com'))
```

2.7 not (notAction, notResource and notPrincipal)

Warning: Make sure, you well understand the concepts of notAction, notResource and notPrincipal. This is where things quickly go wrong, especially when used in combination.

2.7.1 notActions

Switches the policy provider to use NotAction.

TypeScript

```
new statement.S3()
  .allow()
  .notActions()
  .toDeleteBucket()
  .onBucket('some-bucket');
```

JavaScript

```
new statement.S3()
  .allow()
  .notActions()
  .toDeleteBucket()
  .onBucket('some-bucket');
```

Python

```
statement.S3() \
    .allow() \
    .not_actions() \
    .to_delete_bucket() \
    .on_bucket('some-bucket')
```

2.7.2 notResources

Switches the policy provider to use NotResource.

TypeScript

```
new statement.S3()
  .allow()
  .notResources()
  .toDeleteBucket()
  .onBucket('some-bucket');
```

JavaScript

```
new statement.S3()
  .allow()
  .notResources()
  .toDeleteBucket()
  .onBucket('some-bucket');
```

Python

```
statement.S3() \
    .allow() \
    .not_resources() \
    .to_delete_bucket() \
    .on_bucket('some-bucket')
```

2.7.3 notPrincipals

Switches the policy provider to use NotPrincipal.

TypeScript

```
new statement.Sts()
  .deny()
  .notPrincipals()
  .toAssumeRole()
  .forUser('1234567890', 'Bob');
```

JavaScript

```
new statement.Sts()
  .deny()
  .notPrincipals()
  .toAssumeRole()
  .forUser('1234567890', 'Bob');
```

Python

```
statement.Sts() \
    .deny() \
    .not_principals() \
    .to_assume_role() \
    .for_user('1234567890', 'Bob')
```

CHAPTER

THREE

EXAMPLES

Attention: This is an early version of the package. The API might change when new features are implemented. Therefore make sure you use an exact version in your `package.json`/`requirements.txt` before it reaches 1.0.0.

TypeScript

```
const policy = {
  Version: '2012-10-17',
  Statement: [
    new statement.Ec2()
      .allow()
      .toStartInstances()
      .ifAwsRequestTag('Owner', '${aws:username}'),
    new statement.Ec2()
      .allow()
      .toStopInstances()
      .ifResourceTag('Owner', '${aws:username}'),
    new statement.Ec2() //
      .allow()
      .allListActions()
      .allReadActions(),
  ],
};
```

JavaScript

```
const policy = {
  Version: '2012-10-17',
  Statement: [
    new statement.Ec2()
      .allow()
      .toStartInstances()
      .ifAwsRequestTag('Owner', '${aws:username}'),
    new statement.Ec2()
      .allow()
      .toStopInstances()
      .ifResourceTag('Owner', '${aws:username}'),
    new statement.Ec2() //
      .allow()
      .allListActions()
      .allReadActions(),
```

(continues on next page)

(continued from previous page)

```
  ],
};
```

Python

```python
policy = {
    'Version': '2012-10-17',
    'Statement': [
        statement.Ec2()
        .allow()
        .to_start_instances()
        .if_aws_request_tag('Owner', '${aws:username}')
        .to_json(),
        statement.Ec2()
        .allow()
        .to_stop_instances()
        .if_resource_tag('Owner', '${aws:username}')
        .to_json(),
        statement.Ec2()
        .allow()
        .all_list_actions()
        .all_read_actions()
        .to_json()
    ]
}
```

TypeScript

```typescript
const policy = {
  Version: '2012-10-17',
  Statement: [
    new statement.Cloudformation() // allow all CFN actions
      .allow()
      .allActions(),
    new statement.All() // allow absolutely everything that is triggered via CFN
      .allow()
      .allActions()
      .ifAwsCalledVia('cloudformation.amazonaws.com'),
    new statement.S3() // allow access to the CDK staging bucket
      .allow()
      .allActions()
      .on('arn:aws:s3:::cdktoolkit-stagingbucket-*'),
    new statement.Account() // even when triggered via CFN, do not allow␣
↪modifications of the account
      .deny()
      .allPermissionManagementActions()
      .allWriteActions(),
    new statement.Organizations() // even when triggered via CFN, do not allow␣
↪modifications of the organization
      .deny()
      .allPermissionManagementActions()
      .allWriteActions(),
  ],
};
```

JavaScript

```
const policy = {
  Version: '2012-10-17',
  Statement: [
    new statement.Cloudformation() // allow all CFN actions
      .allow()
      .allActions(),
    new statement.All() // allow absolutely everything that is triggered via CFN
      .allow()
      .allActions()
      .ifAwsCalledVia('cloudformation.amazonaws.com'),
    new statement.S3() // allow access to the CDK staging bucket
      .allow()
      .allActions()
      .on('arn:aws:s3:::cdktoolkit-stagingbucket-*'),
    new statement.Account() // even when triggered via CFN, do not allow␣
↪modifications of the account
      .deny()
      .allPermissionManagementActions()
      .allWriteActions(),
    new statement.Organizations() // even when triggered via CFN, do not allow␣
↪modifications of the organization
      .deny()
      .allPermissionManagementActions()
      .allWriteActions(),
  ],
};
```

Python

```
policy = {
    'Version': '2012-10-17',
    'Statement': [
        # allow all CFN actions
        statement.Cloudformation() \
        .allow() \
        .all_actions() \
        .to_json(),
        # allow access to the CDK staging bucket
        statement.All() \
        .allow() \
        .all_actions() \
        .if_aws_called_via('cloudformation.amazonaws.com') \
        .to_json(),
        # allow access to the CDK staging bucket
        statement.S3() \
        .allow() \
        .all_actions() \
        .on('arn:aws:s3:::cdktoolkit-stagingbucket-*') \
        .to_json(),
        # even when triggered via CFN, do not allow modifications of the
        #  account
        statement.Account() \
        .deny() \
        .all_permission_management_actions() \
        .all_write_actions() \
        .to_json(),
        # even when triggered via CFN, do not allow modifications of the
```

(continues on next page)

(continued from previous page)

```
        #  organization
        statement.Organizations() \
        .deny() \
        .all_permission_management_actions() \
        .all_write_actions() \
        .to_json()
    ]
]
```

CHAPTER

FOUR

COLLECTIONS

Attention: This is an early version of the package. The API might change when new features are implemented. Therefore make sure you use an exact version in your `package.json`/`requirements.txt` before it reaches 1.0.0.

IAM Floyd provides commonly used statement collections. These can be called via:

TypeScript

```
new statement.Collection().allowEc2InstanceDeleteByOwner();
```

JavaScript

```
new statement.Collection().allowEc2InstanceDeleteByOwner();
```

Python

```
statement.Collection().allow_ec2_instance_delete_by_owner()
```

Collections return a list of statements, which then can be used in a policy like this:

TypeScript

```
const policy = {
  Version: '2012-10-17',
  Statement: [...new statement.Collection().allowEc2InstanceDeleteByOwner()],
};
```

JavaScript

```
const policy = {
  Version: '2012-10-17',
  Statement: [...new statement.Collection().allowEc2InstanceDeleteByOwner()],
};
```

Python

```
statements = statement.Collection().allow_ec2_instance_delete_by_owner()

policy = {
    'Version': '2012-10-17',
    'Statement': list(map(lambda x: x.to_json(), statements)),
}
```

4.1 Available collections

4.1.1 allowEc2InstanceDeleteByOwner

Allows stopping EC2 instance only for the user who started them.

CHAPTER FIVE

FREQUENTLY ASKED QUESTIONS

Attention: This is an early version of the package. The API might change when new features are implemented. Therefore make sure you use an exact version in your `package.json/requirements.txt` before it reaches 1.0.0.

5.1 Why should I use this package instead of writing policies by hand?

All actions, conditions and resource types of every service are explorable via code suggestion. The related documentation is available in the method description. In most cases you can avoid reading the documentation completely.

IntelliSense makes it super easy to find what you're looking for. But it also helps with discovering things you were not looking for! Users write more secure/restrictive policies because they can easily type `.if` and add conditions with a `<tab>` without looking up multiple documentation pages.

By calling methods of a class you protect yourself against typos. If your code doesn't compile/run because of a typo, you'll immediately notice. If instead you have a typo in your action list, IAM will silently accept your policy. You won't notice until you see a warning in the IAM console.

Allowing/Denying all actions based on access level is a feature AWS missed when designing IAM policies. With this package it is as easy as calling `.allWriteActions()`, `.allReadActions()` etc.

In IAM policies you can use wildcards to add actions to the statement. Wildcards often do not have enough power to define patterns and quickly include too many actions. This package enables you to select actions with regular expressions.

Limiting actions to specific resources via ARN is cumbersome. In this package, for every resource type there is a method, which not only helps with ARN creation - it also adds context to the code which helps to understand the meaning. The classical example here is to allow all actions on an S3 bucket and its containing objects:

```
{
  "Effect": "Allow",
  "Action": "s3:*",
  "Resource": [
    "arn:aws:s3:::example-bucket",
    "arn:aws:s3:::example-bucket/some/path/*"
  ]
}
```

The first resource element is for the bucket itself. The second element is for the contained objects.

A beginner might make the mistake to think the first *or* the last entry is superfluous and remove it. This package has distinct methods to limit actions to a bucket and/or objects:

TypeScript

```
new statement.S3()
  .allow()
  .allActions()
  .onBucket('some-bucket')
  .onObject('some-bucket', 'some/path/*');
```

JavaScript

```
new statement.S3()
  .allow()
  .allActions()
  .onBucket('some-bucket')
  .onObject('some-bucket', 'some/path/*');
```

Python

```
statement.S3() \
    .allow() \
    .all_actions() \
    .on_bucket('some-bucket') \
    .on_object('some-bucket', 'some/path/*')
```

And yes, it's shorter too.

5.2 Are all actions / conditions / resource types covered?

The code of IAM Floyd is generated from the AWS Documentation. This means, **everything that was documented is covered**. Unfortunately not everything is documented. Users have repeatedly reported missing actions/conditions/resource types on the documentation repository.

If you believe something is missing, feel free to report it in the IAM Floyd repository or directly on the AWS Documentation repository.

5.3 How often will there be updates to reflect IAM changes?

Once per hour the AWS Documentation is checked for updates. If anything changes, a new package will be released immediately.

5.4 Do you release new packages when a new CDK version is released?

No. I believe it's a myth and a user error if packages are incompatible with new releases of the CDK. `cdk-iam-floyd` is based on cdk `^1.30.0` and so far I have not seen any issues.

5.5 Is the package following semantic versioning?

Mostly. For manual changes by developers this package follows semver.

Automatic releases triggered by changes in the IAM documentation will always result in a minor update.

It has been observed that IAM actions have been deleted or renamed. This case will not be reflected by a major update! If you had been using such a method your code will break. On the other hand, your code probably already is broken, since it creates a policy with invalid actions until you update to the latest release.

5.6 I don't like method chaining!

That's not a question. But yes, you can completely avoid method chaining:

TypeScript

```
const myStatement = new statement.Ec2();
myStatement.allow();
myStatement.toStartInstances();
myStatement.toStopInstances();
```

JavaScript

```
const myStatement = new statement.Ec2();
myStatement.allow();
myStatement.toStartInstances();
myStatement.toStopInstances();
```

Python

```
my_statement = statement.Ec2()
my_statement.allow()
my_statement.to_start_instances()
my_statement.to_stop_instances()
```

5.7 Floyd?

George Floyd has been murdered by racist police officers on May 25, 2020.

This package is not named after him to just remind you of him and his death. I want this package to be of great help to you and I want you to use it on a daily base. Every time you use it, I want you to remember our society is ill and needs change. The riots will stop. The news will fade. The issue persists!

If this statement annoys you, this package is not for you.

CHAPTER SIX

LEGAL

Attention: This is an early version of the package. The API might change when new features are implemented. Therefore make sure you use an exact version in your `package.json`/`requirements.txt` before it reaches 1.0.0.

The code contained in the lib/generated folder is generated from the AWS documentation. The class- and function-names and their description therefore are property of AWS.

AWS and their services are trademarks, registered trademarks or trade dress of AWS in the U.S. and/or other countries.

This project is not affiliated, funded, or in any way associated with AWS.

6.1 License

IAM Floyd is licensed under Apache License 2.0.

CHAPTER

SEVEN

IAM FLOYD

Attention: This is an early version of the package. The API might change when new features are implemented. Therefore make sure you use an exact version in your `package.json`/`requirements.txt` before it reaches 1.0.0.

AWS IAM policy statement generator with fluent interface.

Support for:

- 241 Services
- 8007 Actions
- 786 Resource Types
- 484 Conditions

7.1 Similar projects

- cdk-iam-actions
- cdk-iam-generator
- iam-policy-generator
- policyuniverse
- policy_sentry