ECS Compose-X Documentation

Release 0.14.3

John Preston

Apr 22, 2022

ECS COMPOSE-X

1 ECS Compose-X 1
2 Requirements 5
3 Installation 7
4 ECS Compose-X as an AWS CloudFormation Macro 9
5 Contributing 13
6 AWS ECS (and AWS Fargate) Features 17
7 Docker Compose 21
8 Docker AWS ECS Plugin 25
9 AWS IAM Policies from AWS SAM 27
10 services 29
11 volumes 31
12 secrets 35
13 networks 39
14 logging 41
15 deploy 43
16 x-scaling 49
17 x-iam 51
18 x-network 53
19 x-logging 57
20 x-xray 59
21 x-codeguru-profiler 61
22 Common syntax for x-resources 65

23 x-dynamodb 69

24 x-rds 73

25 x-docdb 79

26 x-elastic_cache 85

27 x-s3 89

28 x-efs 97

29 x-appmesh 101

30 x-dns 109

31 x-elbv2 111

32 x-acm 119

33 x-kinesis 121

34 x-sqs 125

35 x-sns 131

36 x-events 133

37 x-kms 135

38 x-vpc 139

39 x-cluster 143

40 x-alarms 145

41 spot_config 149

42 Docker ECS Plugin support 151

43 History 155

44 Extras 167

45 Philosophy 175

46 What does ECS Compose-X do differently? Long version 177

47 Why did I create ECS Compose-X? 179

48 Why am I not using AWS CDK? 181

49 Implementing least privileges at the heart of ECS Compose-X 183

50 Contributors 185

51 Credits 187

52 Indices and tables 189

CHAPTER

ONE

ECS COMPOSE-X

1.1 Manage, Configure and deploy your applications/services and AWS resources from your docker-compose definitions

1.1.1 Why use ECS Compose-X?

As a developer, working locally is a crucial part of your day to day work, and **docker-compose** allows you to do just that, for simple services as well as very complex structures.

Your prototype works, and you want to deploy to AWS. But what about IAM ? Networking ? Security ? Configuration ?

Using ECS Compose-X, you keep your docker-compose definitions as they are, add the AWS services you have chosen as part of that definition, such as ELB, RDS/DynamodDB Databases etc, and the program will automatically generate all the AWS CloudFormation templates required to deploy all your services.

It automatically takes care of network access requirements and IAM permissions, following best practices.

1.1.2 Installation

ECS Compose-X can be used as a CLI ran locally, in CICD pipelines, or as an AWS CloudFormation macro, allowing you to use your Docker Compose files directly in CloudFormation!

On AWS using AWS CloudFormation Macro

You can now deploy the CloudFormation macro to your AWS Account using AWS Serverless Application Repository (SAR).

Deploy it in your account today Launch Stack

Find out how to use ECS Compose-X in AWS here

Via pip

```
pip install ecs_composex
```

1.1.3 CLI Usage

```
usage: ecs-compose-x [-h] {up,render,create,config,init,version} ...

positional arguments:
  {up,render,create,config,init,version}
                        Command to execute.
    up                  Generates & Validates the CFN templates,
                        Creates/Updates stack in CFN
    render              Generates & Validates the CFN templates locally. No
                        upload to S3
    create              Generates & Validates the CFN templates locally.
                        Uploads files to S3
    config              Merges docker-compose files to provide with the final
                        compose content version
    init                Initializes your AWS Account with prerequisites
                        settings for ECS
    version             ECS Compose-X Version

optional arguments:
  -h, --help            show this help message and exit
```

Examples

```
# Render all your CFN templates from your docker compose and extension files
ecs-compose-x render --format yaml -n my-awesome-app -f docker-compose.yml -f aws.yml␣
↪-d outputs

# Deploy / Update your application to AWS
ecs-compose-x up --format yaml -n my-awesome-app -f docker-compose.yml -f aws.yml -d␣
↪outputs
```

1.1.4 How is it different ?

There are a lot of similar tools out there, including published by AWS. So here are a few of the features that we think could be of interest to you.

Modularity / “Plug & Play”

The majority of people who are going to use ECS Compose-X on a daily basis should be developers who need to have an environment of their own and want to quickly iterate over it.

However, it is certainly something that Cloud Engineers in charge of the AWS accounts etc. would want to use to make their own lives easy too.

In many areas, you as the end-user of Compose-X will already have infrastructure in place: VPC, DBs and what not. So as much as possible, you will be able in Compose-X to define *Lookup* sections which will find your existing resources, and map these to the services.

Built for AWS Fargate

However the original deployments and work on this project was done using EC2 instances (using SpotFleet), everything is now implemented to work on AWS Fargate First (2020-06-06).

That said, all features that can be supported with EC2 instances are available to you with ECS Compose-X, which, will simply disable such settings when deployed on top of AWS Fargate.

Attributes auto-correct

A fair amount of the time, deployments via AWS CloudFormation, Ansible and other IaC will fail because of incompatible settings. This happened a number of times, with a lot of different AWS Services.

Whilst giving you the ability to use all properties of AWS CloudFormation objects, whenever possible, ECS Compose-X will understand how two services are connected and will auto-correct the settings for you.

For example, if you set the Log retention to be 42 days, which is invalid, it will automatically change that to the closest valid value (here, 30).

1.1.5 Credits

This package would not have been possible without the amazing job done by the AWS CloudFormation team! This package would not have been possible without the amazing community around Troposphere! This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

REQUIREMENTS

2.1 AWS Account configuration

2.1.1 IAM Permissions to execute ECS Compose-X

Since ECS Compose-X adds more and more features, we highly recommend to use the AWS Managed policy **arn:aws:iam:aws::policy/ReadOnlyAccess**.

Additionally, you will need to use all the features and push your files to S3

Listing 1: ECS Compose-X specific permissions

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowFullCloudFormationAccess",
            "Effect": "Allow",
            "Resource": [
                "*"
            ],
            "Action": [
                "cloudformation:*"
            ]
        },
        {
            "Sid": "S3BucketObjectsAccess",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::${BucketName}/*"
            ],
            "Action": [
                "s3:PutObject"
            ]
        },
        {
            "Sid": "S3BucketAccess",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::${BucketName}"
            ],
            "Action": [
                "s3:CreateBucket",
```

(continues on next page)

(continued from previous page)

```
                "s3:ListBucket"
            ]
        }
    ]
}
```

2.1.2 ECS Settings

Because of my adhesion to using the Cloud Provider's tools for monitoring, logging, etc, some features and options are enabled and you would get CloudFormation complain about account level settings not being enabled.

Depending on how you are setting up your AWS account(s) you might have to activate these settings if you haven't already.

Note: It is important that you enable AWS VPC Trunking to allow each service tasks to run within the same SecurityGroup and use the extended number of ENIs per instance. Reference: Container ENI Announcement: AWS VPC mode

ECS Account settings can be found at https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-account-settings.html

- ECS - VPC Trunking
- ECS Extended logs and monitoring

Tip: You can now simply run **ecs-composex init** in order to do all of the following and create your default S3 bucket for your CFN templates

```
ecs-composex init
```

Deploy manually

```
aws ecs put-account-setting-default --name awsvpcTrunking --value enabled
aws ecs put-account-setting-default --name serviceLongArnFormat --value enabled
aws ecs put-account-setting-default --name taskLongArnFormat --value enabled
aws ecs put-account-setting-default --name containerInstanceLongArnFormat --value␣
↪enabled
aws ecs put-account-setting-default --name containerInsights --value enabled
```

Hint: If you want to enable these settings for a specific IAM role you can assume yourself, from CLI you can use *aws ecs put-account-setting* as opposed to *aws ecs put-account-setting-default*

```
aws ecs put-account-setting --name awsvpcTrunking --value enabled
aws ecs put-account-setting --name serviceLongArnFormat --value enabled
aws ecs put-account-setting --name taskLongArnFormat --value enabled
aws ecs put-account-setting --name containerInstanceLongArnFormat --value enabled
aws ecs put-account-setting --name containerInsights --value enabled
```

CHAPTER THREE

INSTALLATION

3.1 Deploy to your AWS Account

Region	Lambda Layer based Macro	Docker based Macro
us-east-1	Launch Stack	Launch Stack
eu-west-1	Launch Stack	Launch Stack

3.2 Stable release

3.2.1 From Pip

To install ECS-Compose-X, run this command in your terminal:

```
$ pip install ecs_composex
```

This is the preferred method to install ECS-Compose-X, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guides you through the process.

3.3 From sources

The sources for ECS-Compose-X can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/lambda-my-aws/ecs_composex
```

Or download the tarball:

```
$ curl -OJL https://github.com/lambda-my-aws/ecs_composex/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER
FOUR

ECS COMPOSE-X AS AN AWS CLOUDFORMATION MACRO

4.1 Deploy to your AWS Account

Region	Lambda Layer based Macro	Docker based Macro
us-east-1	Launch Stack	Launch Stack
eu-west-1	Launch Stack	Launch Stack

4.2 Use with an existing docker-compose file

Say you already have a docker-compose file, and you would like to re-use it as a CloudFormation template. Well you now can, with the CloudFormation macro for ECS Compose-X.

Now, AWS CloudFormation would try to evaluate everything in your current file, which has neither resources, or parameters etc. So this is not a valid CloudFormation template.

For that to work though, all you have to do is add the following lines to your template

```yaml
Transform:
  - compose-x
```

From there, you can deploy your template from the AWS Console or from the CLI, for example, as shown below

```bash
CAPABILITIES="APABILITY_AUTO_EXPAND CAPABILITY_IAM CAPABILITY_NAMED_IAM"
aws cloudformation create-stack --template-body file://merged.yml --capabilities $
 ↪{CAPABILITIES} --stack-name macro-demo
```

Hint: If you have multiple docker-compose files you wish to use, you can either do so via *Use with files stored in AWS S3* or simply merge the multiple YAML files together.

4.3 Use with files stored in AWS S3

If you have multiple files and through CICD or otherwise, and decided to store them in AWS S3, you can then re-use these files directly from there.

```
Fn::Transform:
  Name: compose-x
  Parameters:
    ComposeFiles:
      - s3://files.compose-x.io/docker-compose.yml
      - s3://files.compose-x.io/aws.yml
    BucketName: !Sub cfn-templates-${AWS::Region}-${AWS::AccountId}
```

4.4 Customize to your needs or requirements

The provided templates that will allow you to create the Lambda function for the macro and the macro itself, requires an IAM role. Given all the features supported by ECS Compose-X you might want to customize the IAM permissions of the IAM role assigned to the Lambda function.

The current IAM permissions are permissive to gather any information in the account in order to use the *Lookup** feature.

Using multi-account lookup

If you wish to use the *Lookup* feature, this is totally possible. Simply ensure that your docker-compose file indicates which **RoleArn** to use for the specific lookup and adapt the IAM role of the Lambda function role to allow **sts:AssumeRole** on that role ARN you are indicating.

4.5 CFN Macro Parameters

Listing 1: Parameters syntax reference

```
ComposeFiles: <list>
BucketName: <str>
```

4.5.1 ComposeFiles

The List of files you want to have compiled together in order to deploy your stack

> **Attention:** Just like with the CLI, the order in which the files are composed together (first file least priority, last highest priority) the order you list files in **ComposeFiles** matters in the same way.

4.5.2 BucketName

The name of the Bucket you have allowed the Lambda Function used for the CFN Macro to upload files to.

4.6 Current Limitations

4.6.1 environment files (env_files)

Because of the nature of the syntax requirement for env_files, these are not supported to work with the CFN macro, as the files are not present in the local filesystem.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/compose-x/ecs_composex/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

5.1.4 Write Documentation

ECS-ComposeX could always use more documentation, whether as part of the official ECS-ComposeX docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/lanbda-my-aws/ecs_composex/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *ecs_composex* for local development.

1. Fork the *ecs_composex* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/ecs_composex.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv ecs_composex
$ cd ecs_composex/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make lint
$ make coverage
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests. Use *make coverage* to run both tests and coverage analysis.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst of the module.

5.4 Tips

To run a subset of tests:

```
$ make test
$ make coverage
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

AWS CodeBuild will build and run the tests

CHAPTER

SIX

AWS ECS (AND AWS FARGATE) FEATURES

6.1 Container Definition

Property Name	Supported	Override	Note/Extras
Command	Y	Y	
Cpu	Y	Y	Auto-defined if not set for Fargate
DependsOn	Y	Y	When joined to same family, can depend on each other
DisableNetworking	N	N	N/A
DnsSearchDomains	N	N	Not supported with AWS Fargate
DnsServers	N	N	Not supported with AWS Fargate
DockerLabels	N	Y	Will be added in future version
DockerSecurityOptions	N	N	Not supported with AWS Fargate
EntryPoint	Y	Y	
Environment	Y	Y	
EnvironmentFiles	Y	Y	files automatically copied from local to AWS S3
Essential	Y	Y	Automatically determined based on other deploy labels
ExtraHosts	N	N	Not supported with AWS Fargate
FirelensConfiguration	N	N	
HealthCheck	Y	Y	Full docker-compose support with commands. Separate healcheck with ELBv
Hostname	Y	Y	Disabled with AWS Fargate
Image	Y	Y	
Interactive	N	N	
Links	N	N	Not supported with awsvpc network
LinuxParameters	N	N	
LogConfiguration	Y	Y	Full AWS CloudWatch support
Memory	Y	Y	Auto-defined if not set for Fargate
MemoryReservation			
MountPoints	Y	Y	
Name	Y	Y	Generated by CFN
PortMappings	Y	Y	Full support. Overrides to awsvpc for network
Privileged	N	N	Not supported with AWS Fargate
PseudoTerminal	N	N	
ReadonlyRootFilesystem	N	N	
RepositoryCredentials	Y	Y	
ResourceRequirements	N	N	
Secrets	Y	Y	Strongly automated for RDS and others
StartTimeout	N	N	

Table 1 – continued from previous page

Property Name	Supported	Override	Note/Extras
StopTimeout	N	N	
SystemControls	N	N	
Ulimits	Y	Y	Automatically disable non AWS Fargate supported
User	Y	Y	Expects IDs as docker-compose does
VolumesFrom	N	N	To be implemented
WorkingDirectory	N	N	

6.2 Task Definition

Property Name	Supported	Override	Note/Extras	Compose/X Property
ContainerDefinitions	Y	Y	Strictly generated by Compose-X	services
Cpu	Y	Y	Automatic value for Fargate based on service.resources	deploy.resources *deploy*
ExecutionRoleArn	Y	Y	Strictly generated by Compose-X	*x-iam*
Family	Y	Y	Uses service name or uses label	deploy.labels.ecs.task.family *labels*
InferenceAccelerators	N	N		
IpcMode	N	N		
Memory	Y	Y	**Auto computed for AWS Fargate** based on deploy.resources	deploy.resources
NetworkMode	Y	N	Always awsvpc	
PidMode	N	N	Not supported in Fargate	
PlacementConstraints	N	N	Not applicable to Fargate	
ProxyConfiguration	Y	Y	See x-appmesh	*x-appmesh*
RequiresCompatibilities	Y	N	EC2 and Fargate always defined	
Tags	Y	Y	Generated by Compose-X	See x-tags

6.3 Service Definition

Property Name	Supported	Override	Note/Extras	Compose/X Property
CapacityProviderStrategy	N			
Cluster	Y	Y	x-cluster to create or use	*x-cluster*
DeploymentConfiguration	N			
DeploymentController	Y	N	To date, only ECS	
DesiredCount	Y	N/A		service.deploy.replicas *deploy x-scaling*
EnableECSManagedTags	Y	N		
LoadBalancers	Y	N/A		*x-elbv2*
NetworkConfiguration	Y	Y		service.networks *x-network*
PlacementConstraints	N	N/A		
PlacementStrategies	N	N/A		
PlatformVersion	Y	Y	Default to 1.4.0 for full features support	
PropagateTags	Y	N		
Role	Y	N	Can extend default with x-aws- or x-iam	*x-iam*
SchedulingStrategy	N	N/A		
ServiceArn	N	N/A		
ServiceName	Y	N	Stricly generated by AWS CFN	
ServiceRegistries	Y	Y	See AppMesh	*x-appmesh*
Tags	Y	Y		
TaskDefinition	Y	N	Strictly generated by Compose-X and AWS CFN	

6.4 Cluster definition

All properties for AWS::ECS::Cluster are supported. Pass them through *x-cluster*

CHAPTER

SEVEN

DOCKER COMPOSE

7.1 services

Property Name	Supported	Note/Extras	Replaced By	Reference
build	N			
cap_add	Y			
cap_drop	Y			
command	Y			
configs	N			
cgroup_parent	N			
container_name	Y			
credential_spec	N			
deploy	Y			
devices	N			
depends_on	Y			
dns	N			
dns_search	N			
domainname	N			
tmpfs	N			
entrypoint	Y			
env_file	Y			
environment	Y			
expose	N			
external_links	N			
extra_hosts	N			
group_add	N			
healthcheck	Y			
hostname	N			
image	Y	x-aws-pull_policy supported		
isolation	N			
labels	Y			
links	Y	Ignored when using AWS Fargate		
logging	Y			
network_mode	N	Always set to awsvpc		
networks	Y			
pid	N			
ports	Y	long and short syntax always awsvpc		
secrets	Y	x-secrets		

continues on next page

Table 1 – continued from previous page

Property Name	Supported	Note/Extras	Replaced By	Reference
security_opt	N			
stop_grace_period	N			
stop_signal	N	Incompatible with AWS ECS		
sysctls	Y	Ignored when using AWS Fargate		
ulimits	Y	only nofile for Fargate		
userns_mode	N	Incompatible with AWS ECS		
volumes	Y	x-efs and nfs autodetect		
restart	N	Incompatible with AWS ECS		
shm_size	Y	Ignored when using AWS Fargate		
read_only	Y			
working_dir	Y			

7.1.1 deploy

Tip: See *x-scaling* and *deploy* for more scaling settings. See *labels* for more details on combining services into a single task definition

Hint: Not all ulimits are supported in AWS Fargate. ECS Compose-X Will automatically deactivate the ones not supported.

Tip: **user** expects the format **uid:gid** to use, users and group names aren't supported.

7.2 volumes

Property Name	Supported	Notes/Extras	Replaced By	Reference
driver	Y	nfs autodetect for NFS with AWS EFS		
driver_opts	Y	supports ecs-plugin definition		
driver_opts.type	Y	override to bind for Fargate		
driver_opts.o	N			
driver.name	Y	efs/nfs autodetect for NFS with AWS EFS		
labels	N			
external		Auto defines x-efs.use		
name	Y	Auto defines		

7.3 network

Supported with mapping of AWS VPC & Subnets.

Hint: However DNS features are not supported, you can define a number of DNS Settings for your deployment. See *x-dns*

CHAPTER EIGHT

DOCKER AWS ECS PLUGIN

Property Name	Supported	Compose/X refinement	Reference	Notes
x-aws-cluster	Y	*x-cluster*	*x-aws-cluster* |	
x-aws-pull_credentials	Y		*x-aws-pull_credentials*	
x-aws-autoscaling	Y	*x-scaling*	*x-aws-autoscaling*	
x-aws-policies	Y	*x-iam*	*x-aws-policies*	
x-aws-role	Y	*x-iam*	*x-aws-role*	
x-aws-logs_retention	Y	*x-logging*	*x-aws-logs_retention*	Compose-X Autocorrect to closest valid value
x-aws-min_percent	Y		*x-aws-min_percent & x-aws-max_percent*	
x-aws-max_percent	Y		*x-aws-min_percent & x-aws-max_percent*	

CHAPTER

NINE

AWS IAM POLICIES FROM AWS SAM

ECS Compose-X has defined some IAM permissions for each resource types. In order to provide developers with greater flexibility and use well known system, Compose-X also imports IAM definitions from AWS Serverless Application Model.

You can find all the policies define in AWS SAM in AWS Documentation pages.

9.1 Example

Listing 1: ECS Compose-X Policy for SQS

```yaml
services:
  QueueConsumer: {} # Service definition

x-sqs:
  QueueA:
    Services:
      - name: QueueConsumer
        access: RWMessages
```

Listing 2: Using AWS SAM Policy

```yaml
services:
  QueueConsumer: {} # Service definition

x-sqs:
  QueueA:
    Services:
      - name: QueueConsumer
        access: SQSPollerPolicy
```

In the example above, we are using the **SQSPollerPolicy** which is already defined for us by AWS SAM.

CHAPTER

TEN

SERVICES

We try to re-use as much as possible the docker compose (v3) reference as much as possible.

For the definition of the services, you can simply use the already existing Docker compose definition for your services. Most of the docker-compose services keys are functional, to get a full breakdown, check the *Docker Compose* compatibily matrix.

See also:

Docker Compose 3 file reference

Note: Any property in the docker-compose file you have today, for example, **build** is simply ignored. It will be neither removed nor modified

---

Hint: Checkout the ECS ComposeX secrets definition syntax *secrets* to import AWS Secrets Manager secrets to your container.

CHAPTER

ELEVEN

VOLUMES

This section covers the integration compatibility with docker-compose volumes into AWS ECS.

See also:

docker-compose volumes docker-compose services volumes

11.1 Understand Local volumes vs shared volumes vs persistent volumes

In docker world, one can create docker volumes and attach these to the containers.

As very well synthesized in the tmpfs documentation page, we have

```
Volumes and bind mounts let you share files between the host machine and container so␣
↪that you can persist data even after the container is stopped.

If you’re running Docker on Linux, you have a third option: tmpfs mounts. When you␣
↪create a container with a tmpfs mount, the container can create files outside the␣
↪container’s writable layer.

As opposed to volumes and bind mounts, a tmpfs mount is temporary, and only persisted␣
↪in the host memory. When the container stops, the tmpfs mount is removed, and files␣
↪written there won’t be persisted.
```

In AWS ECS you can use all 3 modes, although, tmpfs is not supported when deploying containers with AWS Fargate, as the host might be shared with other customers, this could create a surface of attack between containers.

Also, it is worth noting that in AWS Fargate, you cannot use the **bind** mounts from the host: again, shared host, this could create a surface of attack from one account to another.

But, that does not mean that in AWS Fargate you cannot create additional volumes outside of your image layers. In fact, AWS Fargate 1.4.0 comes with some encrypted storage for your tasks among other features.

See also:

AWS Fargate 1.4.0 announcement

11.2 Implementation in the AWS + Docker ECS Plugin

The ECS Plugin which allows you to define, in a similar way to ECS Compose-X, your volumes, is of the opinion that any volume you would create is going to be a shared persistent volume using AWS EFS.

As you can see in these examples, you can either leave things by default or define some EFS equivalent properties to define your volumes.

See also:

docker - ecs - volumes syntax reference

11.3 Implementation in ECS Compose-X

To maintain compatibility with the ECS Plugin, if you did specify that the driver should be **nfs** or **efs** (although this is not a supported network driver!), ECS Compose-X will create for you a new FS etc. allowing your containers to connect.

However, by default, ECS Compose-X will follow the behaviour described in the docker-compose volumes reference, which is to respect the **driver** and **driver_opts** settings.

11.3.1 Define a volume for the task only

Although you cannot create a tmpfs in AWS Fargate, you might for consistency with your local development, define a volume just to mount to a specific path.

As per the docker-compose volumes reference, we could have the following

```yaml
services:
  service-01:
    volumes:
      # Just specify a path and let the Engine create a volume
      - /var/lib/mysql
```

There what ECS Compose-X will do is to create in the task definition a new volume using the **local** driver **volume** type, and assign that to the container definition in the task definition specifically.

11.3.2 Define a shared volume between tasks

Alternatively, and this is where the Docker ECS Plugin and ECS Compose-X differ, is in the use of the **volumes** top-level instruction: unless specified otherwise, the volume will be treated as a local but shared volume.

```yaml
volumes:
  shared-volume:

services:
  serviceA:
    volumes:
      - shared-volume:/mnt/shared:rw

  serviceB:
    volumes:
      - source: shared-volume
```

(continues on next page)

(continued from previous page)

```
        target: /mnt/shared
        read_only: false
        type: volume
```

In the above example, we would get a volume created and mounted to both containers.

11.3.3 Define a shared volume using AWS EFS

This is where ECS ComposeX merges back with the Docker ECS Plugin syntax: you can use the same syntax as defined by the Docker ECS Plugin, for example

Using the ECS Plugin syntax reference

```
services:
  test:
    image: my-app
    volumes:
      - db-data:/app/data
volumes:
  db-data:
    driver_opts:
        backup_policy: ENABLED
        lifecycle_policy: AFTER_30_DAYS
        performance_mode: maxIO
        throughput_mode: provisioned
        provisioned_throughput: 1024
```

If you were to use that definition in your compose file with ECS Compose-X, a new EFS will be created with the settings above, along with all the necessary settings for it.

Using the ECS Compose-X specific reference

As usual, you can also define in ECS Compose-X a more comprehensive set of parameters to better define what you want to achieve, using the **x-efs** key.

To go into more details about using **x-efs**, refer to *x-efs*

CHAPTER

TWELVE

SECRETS

As you might have already used these, docker-compose allows you to define secrets to use for the application.

To help continue with docker-compose syntax compatibility, you can now declare your secret in docker-compose, and add an extension field which will be a direct mapping to the secret name you have in AWS Secrets Manager.

ECS ComposeX will automatically add IAM permissions to **the execution** role of your Task definition and will export the secret to your container, using the same name as in the compose file.

See also:

docker-compose secrets reference

Hint: For security purposes, the containers **envoy** and **xray-daemon** are not getting assigned the secrets.

12.1 Syntax

```
x-secrets:
  Name: str
  LinksTo: []
  JsonKeys: []
  Lookup: {}
```

12.1.1 Name

Type: String

The name of the secret in secrets manager to use and import.

Hint: If you want to put the full ARN, you can. There will be a validation for it.

12.1.2 LinksTo

Type: List of Strings

AllowedValues:

- EcsExecutionRole
- EcsTaskRole

If you believe that your service application should have access to the secret via **Task Role**, simply add to the secret definition as follows:

```
secret-name:
  x-secrets:
    Name: String
    LinksTo:
      - EcsExecutionRole
      - EcsTaskRole
```

Warning: If you do not specify **EcsExecutionRole** when specifying **LinksTo** then you will not get the secret exposed to your container via AWS ECS Secrets property of your Container Definition

12.1.3 JsonKeys

Type: List of objects/dicts

Note: Only Fargate 1.4.0+ Platform Version supports secrets JSON Key

Listing 1: JsonKeys objects structure

```
SecretKey: str
VarName: str
Transform: str
```

SecretKey

Name of the JSON Key in your secret.

VarName

The Name of the secret specifically for the secret JSON key

Transform

When you want to transform the original secret key into something else, here are simple transforms.

java_properties

Take a string and replaces all letters to their uppercase version and replaces `.` with `_`

title

Set to uppercase the first letter of every word. **some.properties** becomes **Some.Properties**

capitalize

Changes all letters from lower case to uppercase but does not change anything else.

12.2 Examples

Listing 2: Short example

```
secrets:
  topsecret_info:
    x-secrets:
      Name: /path/to/my/secret

services:
  serviceA:
    secrets:
      - topsecret_info
```

Listing 3: Secret with assignment to Task and Execution Role

```
secrets:
  abcd: {}
  john:
    x-secrets:
      LinksTo:
        - EcsExecutionRole
        - EcsTaskRole
      Name: SFTP/asl-cscs-files-dev
```

Listing 4: Secret Looked up from Tags and Name, also using JsonKeys

```
secrets:
  zyx:
    x-secrets:
      Name: secret/with/kmskey
      Lookup:
        Tags:
          - costcentre: lambda
```

(continues on next page)

(continued from previous page)

```
          - composexdev: "yes"
      JsonKeys:
        - SecretKey: username
          VarName: PSQL_USERNAME
        - SecretKey: password
          VarName: PSQL_PASSWORD
```

Listing 5: Secret with assignment to Task and Execution Role

```
secrets:
  abcd: {}
  john:
    x-secrets:
      LinksTo:
        - EcsExecutionRole
        - EcsTaskRole
      Name: arn:aws:secretsmanager:eu-west-1:123456789012:secret:/secret/abcd
```

CHAPTER THIRTEEN

NETWORKS

In docker-compose one can define diffent subnets which would use different properties, as documented here

This allows you to logically bind services on different networks etc, very useful in many scenarios.

In ECS ComposeX, we have added support to allow you to define these networks and logically associate them with AWS VPC Subnets.

Refer to *x-vpc* for a full review of ECS ComposeX syntax definition for subnets mappings.

You can now define extra subnet groups based on different tags and map them to your services for override when using **Lookup** or **Use**

Listing 1: Extra subnets definition

```
x-vpc:
  Lookup:
    VpcId: {}
      AppSubnets: {}
      StorageSubnets: {}
      PublicSubnets: {}
      Custom01:
        Tags: {}
```

Listing 2: define compose networks and associate to a Subnet category

```
networks:
  custom01:
    x-vpc: Custom01
```

Listing 3: Map a compose defined network to a service

```
services:
  serviceA:
    networks:
      - custom01

  serviceB:
    networks:
      custom01: {}
```

Note: As per docker-compose config, the rendered networks in a service is a map / object. But it also can be a list.

CHAPTER

FOURTEEN

LOGGING

In AWS ECS you can define the log driver in a similar way as you do locally. In ECS Compose-X, default settings will be applied and use awslogs driver by default.

For more information on the docker-compose logging syntax, refer to Docker Compose logging syntax reference

14.1 Supported drivers

Currently, any other driver is ignored and AWS Logs is used by default. This is to guarantee deployment success on AWS ECS with AWS Fargate.

14.1.1 awslogs

Option Name	Required	Notes/Features
awslogs-create-group	False	Compose-X creates a new log group by default
awslogs-region	True	When specified, Compose-X only handles IAM to grant. If not set, defaults to AWS::Region
awslogs-endpoint	False	
awslogs-group	True	Defaults to family name when unset
awslogs-stream-prefix	True	Defaults to service name when unset
awslogs-datetime-format	False	
awslogs-multiline-pattern	False	
mode	False	
max-buffer-size	False	

Hint: To set the log retention period, you can use *x-logging* or **x-aws-logs_retention**

CHAPTER FIFTEEN

DEPLOY

The deploy section allows to set various settings around how the container should be deployed, and what compute resources are required to run the service.

For more details on the deploy, see docker documentation for deploy here

At the moment, all keys are not supported, mostly due to the way Fargate by nature is expecting settings to be.

15.1 resources

The resources allow you to define the CPU/RAM reservations and limits. In AWS ECS, the CPU only has one attribute, so ECS Compose-X will **use the highest value of the two if both set**.

Once the container definitions have been generated, the CPU and RAM requirements are added up together. From there, it will automatically select the closest valid Fargate CPU/RAM combination and set the parameter for the Task.

Important: CPUs should be set between 0.25 and 4 to be valid for Fargate, otherwise you will have an error.

15.1.1 replicas

This setting allows you to define how many tasks should be running for a given service. The value is used to define **MicroserviceCount**.

15.1.2 labels

These labels aren't used for much in native Docker compose as per the documentation. They are only used for the service, but not for the containers themselves. Which is great for us, as we can then leverage that structure to implement a merge of services.

In AWS ECS, a Task definition is a group of one or more containers which are going to be running as a one task. The most usual use-case for this, is with web applications, which need to have a reverse proxy (ie. nginx) in front of the actual application. But also, if you used the *use_xray* option, you realized that ECS ComposeX automatically adds the x-ray-daemon sidecar. Equally, when we implement AppMesh, we will also have another side-car container for this.

So, here is the tag that will allow you to merge your reverse proxy or waf (if you used a WAF in container) fronting your web application:

ecs.task.family

For example, you would have:

```yaml
---
# base file for services with the x-keys for BDD
version: '3.8'
secrets:
  abcd: {}
  john:
    x-secrets:
      LinksTo:
        - EcsExecutionRole
        - EcsTaskRole
      Name: SFTP/asl-cscs-files-dev
  zyx:
    x-secrets:
      Name: secret/with/kmskey
      Lookup:
        Tags:
          - costcentre: lambda
      JsonKeys:
        - VarName: ZYX_TEST
          SecretKey: test
services:
  app01:
    logging:
      driver: awslogs
      options:
        awslogs-group: a-custom-name
        awslogs-create-group: "true"
    sysctls:
      - net.core.somaxconn=2048
      - net.ipv4.tcp_syncookies=1
    cap_add:
      - ALL
#    env_file: ./use-cases/env-files/dummy.env
    deploy:
      update_config:
        failure_action: rollback
      labels:
        ecs.task.family: bignicefamily
      resources:
        reservations:
          cpus: '0.25'
          memory: 1GB
    environment:
      LOGLEVEL: DEBUG
      SHELLY: /bin/bash
      TERMY: screen
    image: nginx
    volumes:
      - type: tmpfs
        target: /tmp
        tmpfs:
          size: 1024
      - normal-vol:/var/tmp/shared
      - some-volume:/var/anotherpath:ro
```

(continues on next page)

(continued from previous page)

```
    links:
      - app03:dateteller
    ports:
      - mode: awsvpc
        protocol: tcp
        published: 5000
        target: 5000
    secrets:
      - zyx
    x-logging:
      RetentionInDays: 42
      CreateLogGroup: False
    x-network:
      is_public: False
      UseCloudmap: True
      Ingress:
        Myself: False
        AwsSources:
          - Type: PrefixList
            Id: pl-6da54004
    x-iam:
      Policies:
        - PolicyName: AllowPublishToCw
          PolicyDocument:
            Statement:
              - Action:
                  - cloudwatch:PutMetricData
                Effect: Allow
                Resource:
                  - '*'
                Sid: AllowPublishMetricsToCw
    x-xray: false
    x-scaling:
      Range: "1-4"
  app02:
    depends_on:
      - app01
      - bignicefamily
#    env_file:
#      - ./use-cases/env-files/dummy.env
    deploy:
      update_config:
        failure_action: pause
      labels:
        ecs.task.family: youtoo
      replicas: 2
      resources:
        reservations:
          cpus: '0.1'
          memory: 64000kB
    environment:
      LOGLEVEL: DEBUG
    healthcheck:
      interval: 1m30s
      timeout: 10s
      start_period: 1h
      retries: 3
```

(continues on next page)

(continued from previous page)

```
      test:
        - CMD
        - curl
        - localhost:5000/ping
    image: nginx
    ports:
      - mode: awsvpc
        protocol: tcp
        published: 5000
        target: 5000
    secrets:
      - zyx
    volumes:
      - source: some-volume
        target: /app/data
        type: volume
    x-iam:
      PermissionsBoundary: arn:aws:iam::aws:policy/AdministratorAccess
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AdministratorAccess
    x-scaling:
      Range: "1-5"
      TargetScaling:
        CpuTarget: 88
        DisableScaleIn: true
    x-xray: false
    tmpfs: /run
  app03:
    tmpfs:
      - /run
      - /tmp
    sysctls:
      net.core.somaxconn: 1024
      net.ipv4.tcp_syncookies: 0
    cap_add:
      - NET_ADMIN
      - SYS_PTRACE
    cap_drop:
      - SYS_ADMIN
    ulimits:
      nofile:
        soft: 1024
        hard: 2048
      nproc: 512
    x-aws-min_percent: 50
    x-aws-max_percent: 150
    deploy:
      resources:
        reservations:
          cpus: '0.25'
          memory: 134217728b
    environment:
      LOGLEVEL: DEBUG
    image: nginx
    ports:
      - mode: awsvpc
        protocol: tcp
```

(continues on next page)

(continued from previous page)

```yaml
        published: 5000
        target: 5000
    secrets:
      - abcd
      - zyx
      - john
    volumes:
      - /generated/volume/from/path
      - shared-images:/app/images
      - some-volume:/app/data:ro
    x-network:
      Ingress:
        Myself: False
        ExtSources:
          - Ipv4: 0.0.0.0/0
            Description: ANYWHERE

    x-logging:
        RetentionInDays: 30
    x-scaling:
      Range: 1-10
  rproxy:
    logging:
      driver: awslogs
      options:
        awslogs-region: us-east-1
    depends_on:
      - app01
      - app02
    deploy:
      labels:
        ecs.task.family: bignicefamily,youtoo
      replicas: 1
      resources:
        limits:
          cpus: '0.25'
          memory: 64M
        reservations:
          cpus: '0.1'
          memory: 32M
    image: nginx
    volumes:
      - normal-vol:/tmp/shared
    ports:
      - mode: awsvpc
        protocol: tcp
        published: 80
        target: 80
    x-iam:
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/ReadOnlyAccess
    x-xray: true
    x-network:
      is_public: False
      UseCloudmap: True

volumes:
```

(continues on next page)

(continued from previous page)

```
  shared-images: {}
  some-volume: {}
  normal-vol: {}


x-dns:
  PrivateNamespace:
    Name: lambda.internal

x-tags:
  costcentre: lambda
```

Warning: The example above illustrates that you can either use, for deploy labels

- a list of strings
- a dictionary

ecs.depends.condition

This label allows to define what condition should this service be monitored under by ECS. Useful when container is set as a dependency to another.

Hint: Allowed values are : START, SUCCESS, COMPLETE, HEALTHY. By default, sets to START, and if you defined **healthcheck**, defaults to HEALTHY. See Dependency reference for more information

CHAPTER

SIXTEEN

X-SCALING

Contents

- *x-scaling*
  - *Range*
  - *TargetScaling*
    - *CpuTarget / RamTarget*
    - *ScaleInCooldown / ScaleOutCooldown*
    - *DisableScaleIn*

This section allows to define scaling for the ECS Service. For SQS Based scaling using step scaling, refer to SQS Documentation.

```
services:
  serviceA:
    x-scaling:
      Range: "1-10"
      TargetScaling:
        CpuTarget: 80
```

16.1 Range

Range, defines the minimum and maximum number of containers you will have running in the cluster.

```
#Syntax
# Range: "<min>-<max>"
# Example
Range: "1-21"
```

16.2 TargetScaling

Allows you to define target scaling for the service based on CPU/RAM.

Listing 1: target scaling syntax reference

```
x-scaling:
  Range: "1-10"
  TargetScaling:
    CpuTarget: int (will be casted to float)
    MemoryTarget: int (will be casted to float)
    ScaleInCooldown: int (ie. 60)
    ScaleOutCooldown: int (ie. 60)
    DisableScaleIn: boolean (True/False)
```

16.2.1 CpuTarget / RamTarget

Defines the CPU **percentage** that we want the service to be under. ECS will automatically create and adapt alarms to scale the service in/out so long as the average CPU usage remains beneath that value.

Attention: Note that setting both should not be set at the same time, as you might end up into a racing condition.

16.2.2 ScaleInCooldown / ScaleOutCooldown

This allows you to define the Cooldown between scaling activities in order to limit drastic changes.

Hint: These are set only for the CPU and RAM targets, no impact on other scaling such as SQS.

16.2.3 DisableScaleIn

Default: False

Same as the original Property in the CFN definition, this will deny a service to scale in after it has scaled-out for applications that do not support to scale-in.

Hint: If you define multiple services within the same **family**, the lowest value for CPU/RAM and highest for scale in/out are used in order to minimize the impact and focus on the weakest point.

CHAPTER SEVENTEEN

X-IAM

Contents

- *x-iam*
  - *PermissionsBoundary*
  - *Policies*
  - *ManagedPolicies*

This section is the entrypoint to further extension of IAM definition for the IAM roles created throughout.

17.1 PermissionsBoundary

This key represents an IAM policy (name or ARN) that needs to be added to the IAM roles in order to represent the IAM Permissions Boundary.

Note: You can either provide a full policy arn, or just the name of your policy. The validation regexp is:

```
r"((^([a-zA-Z0-9-_.\/]+)$)|(^(arn:aws:iam::(aws|[0-9]{12}):policy\/)[a-zA-Z0-9-_.\/]+
↪$))"
```

Examples:

```yaml
services:
  serviceA:
    image: nginx
    x-configs:
      iam:
        boundary: containers
  serviceB:
    image: redis
    x-configs:
      iam:
        boundary: arn:aws:iam::aws:policy/PowerUserAccess
```

Tip: if you specify ony the name, ie. **containers**, this will resolve into

arn:${AWS::Partition}:iam::${AWS::AccountId}:policy/containers

17.2 Policies

Allows you to define additional IAM policies. Follows the same pattern as CFN IAM Policies

```yaml
x-iam:
  Policies:
      - PolicyName: somenewpolicy
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Action:
                - ec2:Describe*
              Resource:
                - "*"
              Sid: "AllowDescribeAll"
```

Tip: If you used the ECS Plugin from docker before, this is equivalent to *x-aws-role*

17.3 ManagedPolicies

Allows you to add additional managed policies. You can specify the full ARN or just a string for the name / path of the policy. If will resolve into the same regexp as for *PermissionsBoundary*

Tip: If you used the ECS Plugin from docker before, this is equivalent to *x-aws-policies*

Hint: You can also use the Docker ECS-Plugin **x-aws-iam** extension fields with ECS ComposeX

CHAPTER
EIGHTEEN

X-NETWORK

Listing 1: Overview

```
UseCloudmap: bool
Ingress: {ingress_definition}
```

Contents

- *x-network*
  - *UseCloudmap*
  - *Ingress definition*
    - *Syntax reference*
  - *Map VPC subnets to docker-compose networks*

18.1 UseCloudmap

Boolean to turn on or off the integration to CloudMap for the services.

Default: False

Note: If you want to use appmesh and define **x-appmesh** in the template, automatically, all services will be registered in AWS CloudMap.

18.2 Ingress definition

This allows you to define specific ingress control from external sources to your environment. For example, if you have to whitelist IP addresses that are to be allowed communication to the services, you can list these, and indicate their name which will be shown in the EC2 security group description of the ingress rule.

18.2.1 Syntax reference

```
Ingress:
  ExtSources: []
  AwsSources: []
  Myself: True/False
```

Listing 2: Ingress Example

```
services:
  app01:
    x-network:
      Ingress:
        ExtSources:
          - IPv4: 0.0.0.0/0
            Name: all
          - IPv4: 1.1.1.1/32
            Source_name: CloudFlareDNS
        AwsSources:
          - Type: SecurityGroup
            Id: sg-abcd
          - Type: PrefixList
            Id: pl-abcd
        Myself: True/False
```

Note: Future feature is to allow to input a security group ID and the remote account ID to allow ingress traffic from a security group owned by another of your account (or 3rd party).

Hint: The protocol is automatically detected based on the port definition. By default, it is TCP

Hint: To see details about the Ingress for Load Balancers, refer to *Ingress*

Hint: When using an ALB, you do not need to define that ALB security group etc., all inbound rules will be defined automatically to allow the ALB to communicate with your service!

18.3 Map VPC subnets to docker-compose networks

Listing 3: AWS VPC to network mapping

```
networks:
  internal:
    x-vpc: InteralCustomSubnets

x-vpc:
  VpcId:
    Tags: []
  AppSubnets:
```

(continues on next page)

(continued from previous page)

```
    Tags: []
  PublicSubnets:
    Tags: []
  StorageSubnets:
    Tags: []
  InteralCustomSubnets:
    Tags: []

services:
  serviceA:
    networks: [internal]
```

In some cases, you might have complex VPC topology and created new specific Subnets in **x-vpc**, and map that subnet name to a docker-network defined network. Then later, you can set your service in the services definition to be put into that network.

CHAPTER

NINETEEN

X-LOGGING

The following parameter is identical in behaviour to **x-aws-logs_retention** defined in the docker ECS Plugin.

Listing 1: x-logging syntax definition

```
RetentionInDays: int
```

Hint: Alternatively you can use the ECS Plugin logging definition will ECS Compose-X will use. If both are defined, priority goes to the highest value.

19.1 RetentionInDays

Value to indicate how long should the logs be retained for the service.

Hint: If the value you enter is not in the allowed values, will set to the closest accepted value.

Hint: Emulates the CW Logs property RetentionInDays Property

19.2 Examples

```
services:
  serviceA:
    x-logging:
      RetentionInDays: 42
```

CHAPTER

TWENTY

X-XRAY

This section allows to enable X-Ray to run right next to your container. It will use the AWS original image for X-Ray Daemon and exposes the ports to the task.

20.1 Syntax reference

```
x-xray: True/False
```

20.2 Example

Listing 1: Enable XRay for your service.

```
services:
  serviceA:
    x-xray: True
```

See also:

ecs_composex.ecs.ecs_service#set_xray

20.3 IAM permissions

Enabling XRay will automatically add the following managed policy to your task definition:

arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess

Listing 2: IAM policy definition

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
```

(continues on next page)

(continued from previous page)

```
                "xray:GetSamplingStatisticSummaries"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

CHAPTER

TWENTYONE

X-CODEGURU-PROFILER

Enables to use or create an existing/a new CodeProfiling group for your service.

Unlike most of the resources attachments, this is not done at the "family" level but at the service level, as it might not be wanted to profile every single container in the task.

x-codeguru-profiler is a service/task level setting which offers a 1:1 mapping between your application and the profiler.

Hint: Using ECS ComposeX, this automatically adds an Environment variable to your container, **AWS_CODEGURU_PROFILER_GROUP_ARN** and **AWS_CODEGURU_PROFILER_GROUP_NAME** with the ARN of the newly created Profiling Group.

21.1 Syntax reference / Examples

I wanted to make it easy for people to use as well as being flexible and support all CFN definition.

Listing 1: Syntax for setting pre-defined codeprifiling group without creating a new one.

```
x-codeguru-profiler: name (str)
```

Listing 2: Create a new CodeProfiling group with default settings.

```
x-codeguru-profiler: True|False (bool)
```

Listing 3: Properties as defined in AWS CFN for ProflingGroup

```
x-codeguru-profiler:
  AgentPermissions: Json
  AnomalyDetectionNotificationConfiguration:
    - Channel
  ComputePlatform: String
  ProfilingGroupName: String
  Tags:
    - Tag
```

See also:

AWS CFN definition for CodeGuru profiling group

Note: When you define the properties, in case you already had principals, it will still automatically add the **IAM Task Role** to the list of Principals that should publish to the profiling group.

21.2 Code Example

Here is an example of a simple Flask application I added the codeguru profiler for.

```python
import boto3
import logging
from aws_xray_sdk.ext.flask.middleware import XRayMiddleware
from aws_xray_sdk.core import patcher, xray_recorder
from werkzeug.middleware.proxy_fix import ProxyFix
from codeguru_profiler_agent import Profiler
from app02 import APP


def start_app():
    debug = False
    if "DEBUG" in APP.config and APP.config["DEBUG"]:
        debug = True

    if "USE_XRAY" in APP.config and APP.config["USE_XRAY"]:
        xray_recorder.configure(service=APP.name)
        XRayMiddleware(APP, xray_recorder)
        xray_recorder.configure(service="app01")
        if "USE_XRAY" in APP.config and APP.config["USE_XRAY"]:
            patcher.patch(
                (
                    "requests",
                    "boto3",
                )
            )
        print("Using XRay")

    if APP.config["AWS_CODEGURU_PROFILER_GROUP_NAME"]:
        p = Profiler(
            profiling_group_name=APP.config["AWS_CODEGURU_PROFILER_GROUP_NAME"],
            aws_session=boto3.session.Session(),
        )
        p.start()
        print(
            f"Started profiler {p} for {APP.config['AWS_CODEGURU_PROFILER_GROUP_NAME
↪']}"
        )
        logging.getLogger('codeguru_profiler_agent').setLevel(logging.INFO)

    APP.wsgi_app = ProxyFix(APP.wsgi_app)
    APP.run(host="0.0.0.0", debug=debug)


if __name__ == "__main__":
    start_app()
```

See also:

Full Applications code used for this sort of testing can be found here

CHAPTER

TWENTYTWO

COMMON SYNTAX FOR X-RESOURCES

ECS ComposeX requires to expands onto the original Docker compose file defintion in order to map the docker compose properties to their equivalent settings on AWS ECS and otherwise for the other "Extra" resources.

In general for each x- section of the docker compose document, we will find three attributes to each resource:

- *Properties*
- *Settings*
- *Services*
- *Lookup*

22.1 Properties

Unless indicated otherwise, these are the properties for the resource as you would define them using the AWS properties in the AWS CloudFormation resource definition.

> **Warning:** In order to update some resources, AWS Sometimes needs to create new ones to replace the once already in place, depending on the type of property you are changing. To do so, AWS will need to have the name of the resource generated, and not set specifically for it. It is a limitation, but in the case of most of the resources, it also allows for continued availability of the service to the resources.
>
> Therefore, some resources will not be using the *Name* value that you give to it, if you did so.

22.2 Lookup

Allows you to Lookup existing resources (tagged) that you would like to use with the new services you are deploying. Everything with regards to the access and other properties, depending on the type of resources, will remain the same.

This is accomplished by using AWS Resources Group Tags API which means, at this point in time, you can only find resources that are tagged.

Listing 1: Generic format for Lookup

```yaml
Lookup:
  Tags:
    - Key: Value
    - Key: Value
  RoleArn: <str|optional>
```

22.2.1 Tags

The tags are a list of Tags that have been assigned to the resource. Based on the type of resource, this might need to resolve to a single specific resource in your AWS account / region.

22.2.2 RoleArn

This allows you to provide the ARN of an IAM Role that ComposeX can use in order to lookup for resources. It is very useful in case you plan to do cross-account lookup for shared resources or simply to render your templates in a central CICD account.

Note: It will never modify the looked up object!

Warning: You can only lookup tagged resource on AWS.

Tip: Tags keys and values are case sensitive. At this stage, this does not support regexps.

22.3 Settings

The settings is the section where we can take shortcuts or wrap around settings which would otherwise be complex to define. Sometimes, it simply is an easy way to use helpers which are configurable. For example, in the next interation for the x-rds resources, we will allow to define the latest RDS engine and version that supports Serverless for aurora.

There is a set of settings which are going to be generic to all modules.

22.3.1 EnvNames

Multiple teams who would want to adopt ECS ComposeX might already have their own environment variable keys (or names) for a common resource. For example, team A and team B can use the same SQS queue but they did not define a common name for it, so team A calls it *QueueA* and team B calls it *QUEUE_A*.

With EnvNames, you can define a list of environment variables that will all share the same value, simply have a different name.

Hint: No need to add the name of the resource as defined in the docker compose file, this will always be added by default.

22.3.2 Subnets

Listing 2: Example of override for RDS

```yaml
x-rds:
  dbA:
    Settings:
      Subnets: AppSubnets
```

This parameter allows you to override which subnets should be used for the resource to be deployed to. It applies to that resource only so if you had for example, multiple RDS instances, default behaviour is observed for all resources that do not have this override.

Note: This only applies to services using TCP, so * x-rds * x-docdb * x-elasticache

Note: For ECS services to be deployed into different subnets, refer to *networks*

22.4 Services

This is a list of object, with two keys: name, access. The name points to the service as defined in the docker compose file.

Warning: This is case sensitive and so the name of the service in the list must be the same name as the service defined.

Note: At this point in time, each x- section has its own pre-defined IAM permissions for services that support IAM access to the resources. In a future version, I might add a configuration file to override that behaviour.

Refer to each x- resource syntax to see which access types are available.

CHAPTER TWENTYTHREE

X-DYNAMODB

Listing 1: Syntax reference

```
x-dynamodb:
  table-A:
    Properties: {}
    MacroParameters: {}
    Settings: {}
    Services: []
```

23.1 Properties

Refer to AWS CFN Dynamodb Documentation. We support all of the definition and test with the documentation examples.

Listing 2: Tables with GSI

```
---
# Blog applications

version: '3.8'

x-dynamodb:
  tableA:
    Properties:
      AttributeDefinitions:
        - AttributeName: "Album"
          AttributeType: "S"
        - AttributeName: "Artist"
          AttributeType: "S"
        - AttributeName: "Sales"
          AttributeType: "N"
        - AttributeName: "NumberOfSongs"
          AttributeType: "N"
      KeySchema:
        - AttributeName: "Album"
          KeyType: "HASH"
        - AttributeName: "Artist"
          KeyType: "RANGE"
      ProvisionedThroughput:
        ReadCapacityUnits: "5"
        WriteCapacityUnits: "5"
```

(continues on next page)

(continued from previous page)

```
      GlobalSecondaryIndexes:
        - IndexName: "myGSI"
          KeySchema:
            - AttributeName: "Sales"
              KeyType: "HASH"
            - AttributeName: "Artist"
              KeyType: "RANGE"
          Projection:
            NonKeyAttributes:
              - "Album"
              - "NumberOfSongs"
            ProjectionType: "INCLUDE"
          ProvisionedThroughput:
            ReadCapacityUnits: "5"
            WriteCapacityUnits: "5"
        - IndexName: "myGSI2"
          KeySchema:
            - AttributeName: "NumberOfSongs"
              KeyType: "HASH"
            - AttributeName: "Sales"
              KeyType: "RANGE"
          Projection:
            NonKeyAttributes:
              - "Album"
              - "Artist"
            ProjectionType: "INCLUDE"
          ProvisionedThroughput:
            ReadCapacityUnits: "5"
            WriteCapacityUnits: "5"
      LocalSecondaryIndexes:
        - IndexName: "myLSI"
          KeySchema:
            - AttributeName: "Album"
              KeyType: "HASH"
            - AttributeName: "Sales"
              KeyType: "RANGE"
          Projection:
            NonKeyAttributes:
              - "Artist"
              - "NumberOfSongs"
            ProjectionType: "INCLUDE"

    Services:
      - name: app03
        access: RW
      - name: app02
        access: RW
      - name: bignicefamily
        access: RO
```

23.2 Settings

See the *Settings* for more details.

Hint: Given DynamoDB is serverless (unless using DAX), there is no **Subnets** override.

23.3 Lookup

For more details, see the *Lookup*.

Listing 3: Lookup DynamoDB Table example

```
x-dynamodb:
  table-A:
    Lookup:
      Tags:
        - table-name: table123
        - owner: myself
        - costallocation: 123
    Services:
      - name: serviceA
        access: DynamoDBCrudPolicy
```

ECS Compose-X defined access names:

- RW : Allow read/write/delete on the table items
- RO: Allow read only actions on the table items

Some of the AWS SAM access:

- DynamoDBCrudPolicy
- DynamoDBReadPolicy
- DynamoDBWritePolicy

23.4 Services

Listing 4: Define services

```
Services:
  - name: serviceA
    access: RW
  - name: serviceB
    access: RO
```

CHAPTER

TWENTYFOUR

X-RDS

24.1 Syntax

```
x-rds:
  psql-dbA:
    Properties: {}
    MacroParameters: {}
    Settings: {}
    Services: []
    Lookup: {}
```

24.2 Properties

RDS clusters or instances need a lot of properties. In order to keep compatibility you can still provide all the properties that the RDS Cluster or RDS Instance would need with the same definition as in AWS CloudFormation.

However, some settings will be replaced automatically (at least for the foreseeable future), such as the master username and password. The reason for it is to allow to keep integration to your ECS Services as seamless as possible.

24.2.1 Using properties

When using Properties, you can use either the RDS Aurora Cluster properties or RDS Instances properties. ECS ComposeX will attempt to automatically identify whether this is a DB Cluster or DB Instance properties set. If successful, it will ingest all your properties, and explained earlier, interpolate a few with new ones created for you.

- MasterUsername
- MasterUserPassword
- Security Groups

24.3 MacroParameters

MacroParameters for RDS allow you to set only very little settings / properties and let ECS ComposeX do the rest for you.

Listing 1: MacroParameters syntax

```
Engine: str
EngineVersion: str
UseServerless: bool
UseMultiAz: bool
ParametersGroup: {}         # Properties for parameters group as per AWS CFN␣
↪definition
Instances: []               # Only valid when creating a DBCluster, allows to define␣
↪multiple DB Instances
RdsFeatures: {}             # Custom settings to define AWS RDS AssociatedRoles
PermissionsBoundary: str    # Allow you to define an IAM boundary policy that will be␣
↪used for the RDS IAM role(s)
```

Listing 2: MacroParameters definitions example

```
Engine: aurora-postgresql # Same as AWS CFN Engine property
EngineVersion: 11.7 # Same as AWS CFN EngineVersion property
UseServerless: False
UseMultiAz: True
ParametersGroups:
  Description: Some description
  Family: aurora-postgresql-11.7
  Parameters: {}
Instances: []
RdsFeatures:
  - Name: s3Import
    Resources:
      - x-s3::bucket-01
      - arn:aws:s3:::bucket/path/allowed/*
      - bucket-name
```

24.3.1 PermissionsBoundary

Allows to define whether an IAM Policy boundary is required for the IAM roles that will be created around the RDS Cluster/Instance.

Hint: This value can be either a policy name or policy ARN. When a policy Name, the ARN is built based on your Account ID.

24.3.2 RdsFeatures

Listing 3: Syntax definition

```
RdsFeatures:
  - Name: <DB Engine feature name>
  - Resources: [<str>]
```

The RDS Features is a wrapper to automatically define which RDS Features, supported by the Engine family, you might want to enable. For these features, which require an IAM role, it will create a new IAM role specifically linked to RDS and grant permissions based on the what the feature requires.

If you had set **AssociatedRoles** already in the permissions, then each *FeatureName* you have already defined that you might re-define in **RdsFeatures** will be skipped. If you wish to use **RdsFeatures** then remove that feature from the **AssociateRoles** definition.

Attention: This was primarily developed to allow feature request #375 so at the moment it only supports s3Import and s3Export.

Listing 4: Example with different bucket names syntax

```
x-rds:
  dbB:
    Properties: {}
    MacroParameters:
      PermissionsBoundary: policy-name
      RdsFeatures:
        - Name: s3Import
          Resources:
            - x-s3::bucket-01
            - arn:aws:s3:::sacrificial-lamb/folder/*
            - bucket-name
        - Name: s3Export
          Resources:
            - x-s3::bucket-01
            - arn:aws:s3:::sacrificial-lamb/folder/*
            - bucket-name
```

Hint: You can reference a S3 bucket defined in **x-s3**. This supports S3 buckets created and referenced via Lookup

24.4 Services

At this point in time, there is no plan to deploy as part of ECS ComposeX a lambda function that would connect to the DB and create a DB/schema specifically for the microservice, as would this lambda function do.

The syntax for listing the services remains the same as the other x- resources but the access type won't be respected.

24.4.1 Access types

Warning: The access key value won't be respected at this stage. This is required to keep compatibility with other modules.

24.5 Settings

Listing 5: Supported Settings

```
EnvNames: [<str>] # List of Environment Variable names to use for exposure to␣
↪container
```

24.6 Lookup

The lookup allows you to find your cluster or db instance and also the Secret associated with them to allow ECS Services to get access to these.

It will also find the DB security group and add an ingress rule.

```
x-rds:
  dba:
    Lookup:
      cluster:
        Name: cluster-identifier
        Tags:
          - sometag: value
      instance:
        Name: DB Instance Id
        Tags:
          - sometag: value
      secret:
        Tags:
          - sometag: value
        Name: secret/in/secretsmanager
```

When using AWS RDS Aurora, you should be specifying the cluster, otherwise the instance for "traditional" RDS instances.

24.7 Defaults

24.7.1 Credentials

Aurora and traditional RDS Databases support both Username/Password generic authentication. Due to the wide adoption of that authentication mechanism, all RDS Dbs will come with a username/password, auto generated and stored in AWS Secrets Manager.

Hint: We do plan to allow a tick button to enable Aurora authentication with IAM, however have not received a Feature Request for it.

AWS Secrets Manager integrates very nicely to AWS RDS. This has no intention to implement the rotation system at this point in time, however, it will generate the password for the database and expose it securely to the microservices which can via environment variables fetch

- DB Endpoint
- DB username
- DB Password
- DB Port

24.8 Examples

Listing 6: New DB Creation

```
x-rds:
  dbname:
    Properties:
      Engine: aurora-mysql
      EngineVersion: 5.7.12
    Services:
      - name: app01
        access: RW
```

Listing 7: Existing Cluster DB Lookup

```
x-rds:
  existing-cluster-dbA:
    Lookup:
      cluster:
        Tags:
          - key: value
      secret:
        Tags:
          - key: value
```

Hint: The DB Family group will be found automatically and the setting will allow creation of a new RDS Parameter group for the Cluster / DB Instance.

CHAPTER

TWENTYFIVE

X-DOCDB

25.1 Syntax

```
x-docdb:
  docdb-01:
    Properties: {}
    Settings: {}
    Services: []
    Lookup: {}
    MacroParameters: {}
```

Tip: For production workloads, to avoid any CFN deadlock situations, I recommend you generate the CFN templates for docdb, and deploy the stacks separately. Using Lookup you can use existing DocDB clusters with your new services.

25.2 Properties

DocDB Cluster is rather very simple in its configuration. There aren't 200+ combinations of EngineName and Engine Version as for RDS, make life very easy.

However you can copy-paste all the properties you would find in the DocDB Cluster properties, some properties will be ignored in order to keep the automation going:

- **MasterUsername and MasterUserPassword** These two will be auto generated and stored in secrets manager. The services linked to it will be granted **GetSecretValue** to it.
- **VpcSecurityGroupIds** The security group will be generated for the DB specifically and allow services listed only.
- **AvailabilityZones** Under trial, but not sure given that we give a Subnet Group why one would also define the AZs and it might conflict.
- **DBClusterIdentifier** As usual, named resources make for a nightmare to rename etc. Instead, there will be a **Name** tag associated with your Cluster.
- **DBSubnetGroupName** Equally gets created only. For now.
- **SnapshotIdentifier** Untested - 2020-11-13 - will support it later.

25.3 MacroParameters

These parameters will allow you to define extra parameters to define your cluster successfully.

```
Instances: []
DBClusterParameterGroup: {} # AWS DocDB::DBClusterParameterGroup properties
```

25.3.1 Instances

List of DocDB instances. The aspiration is to follow the same syntax as the DocDB Instance.

Note: Not all Properties are respected, instead, they follow logically the attachment to the DocDB Cluster.

```
Instances:
  - DBInstanceClass: <db instance type>
    PreferredMaintenanceWindow: <window definition>
    AutoMinorVersionUpgrade: bool
```

Hint: If you do not define an instance, ECS ComposeX automatically creates a new one with a single node of type **db.t3.medium**

25.3.2 DBClusterParameterGroup

Allows you to create on-the-fly parameter groups to tune your DocDB cluster. Refer to DocDB DBClusterParameterGroup for more details.

Listing 1: parameter groups example

```
Description: "description"
Family: "docdb3.6"
Name: "sampleParameterGroup"
Parameters:
  audit_logs: "disabled"
  tls: "enabled"
  ttl_monitor: "enabled"
```

25.4 Services

The syntax for listing the services remains the same as the other x- resources.

```
Services:
  - name: <service/family name>
    access: <str>
```

25.4.1 Access types

Warning: The access key value do not have an effect at this stage.

25.5 Settings

The only setting for DocumentDB is **EnvNames** as for every other resources.

Hint: Given that the DB Secret attachment populates host, port etc., we expose as env vars the **Secret** associated to the DB, not the DB itself.

25.6 Lookup

Lookup for Document DB is available!

Warning: For some reason the group resource tag API returns two different clusters even though they are the same one. Make sure to specify the *Name* along with Tags until we figure an alternative solution. Sorry for the inconvenience.

25.7 Credentials

The credentials strucutre remains the same as for RDS SQL versions

Listing 2: DocumentDB secret structure after attachment

```
{
  "dbClusterIdentifier": "<str>",
  "password": "<str>",
  "engine": "<str>",
  "port": "<int>",
  "host": "<str>",
  "username": "<str>"
}
```

25.8 Examples

Listing 3: Sample to crate two DBs with different instances configuration

```
---
# DOCDB Simple use-case. Creating new DBs


x-docdb:
  docdbA:
```

(continues on next page)

(continued from previous page)

```yaml
    Properties: {}
    Settings:
      EnvNames:
        - DOCDB_A
    Services:
      - name: app03
        access: RW

  docdbB:
    Properties: {}
    Settings:
      EnvNames:
        - DOCDB_B
    Services:
      - name: app03
        access: RW
    MacroParameters:
      Instances:
        - DBInstanceClass: db.r5.large
        - DBInstanceClass: db.r5.xlarge
          AutoMinorVersionUpgrade: True

  docdbC:
    Properties:
      BackupRetentionPeriod: 7
      DBSubnetGroupName: String
      DeletionProtection: False
      EngineVersion: 4.0.0
      StorageEncrypted: True
      Tags:
        - Key: Name
          Value: docdb_C
    Services:
      - name: app03
        access: RW
    MacroParameters:
      Instances:
        - DBInstanceClass: db.r5.large
        - DBInstanceClass: db.t3.medium
          AutoMinorVersionUpgrade: True
      DBClusterParameterGroup:
        Description: "Some description"
        Family: "docdb4.0"
        Name: "sampleParameterGroup"
        Parameters:
          audit_logs: "disabled"
          tls: "disabled"
          ttl_monitor: "enabled"
```

Listing 4: Create a DocDB and import an existing one.

```yaml
---
# DOCDB Simple use-case. Creating new DBs


x-docdb:
```

(continues on next page)

(continued from previous page)

```
  docdbA:
    Properties: {}
    Settings:
      EnvNames:
        - DOCDB_A
    Services:
      - name: app03
        access: RW

  docdbB:
    Settings:
      EnvNames:
        - DOCDB_A
    Services:
      - name: app03
        access: RW
    Lookup:
      cluster:
        Name: docdbb-purmjgtgvyqr
        Tags:
          - CreatedByComposeX: "true"
          - Name: docdb.docdbB
      secret:
        Tags:
          - aws:cloudformation:logical-id: docdbBSecret
```

CHAPTER

TWENTYSIX

X-ELASTIC_CACHE

Listing 1: syntax reference

```
Properties: {}       # AWS CacheCluster or ReplicationGroup properties
MacroParameters: {} # Shortcut parameters to get going quickly
Settings: {}         # Generic settings supported by all resources
Services: []         # List of services that will get automatically access to the␣
↪resource.
Lookup: {}           # Lookup definition to find existing Cache or ReplicationGroup.
```

Hint: ECS ComposeX will always create a new SecurityGroup for a new resource to ensure the services can get access by setting EC2 Security Ingress rules.

26.1 Properties

This allows you to define all the properties for either the AWS CacheCluster or AWS Replication Group resource as part of the AWS ElasticCache family.

ECS ComposeX will automatically detect which of the two resource it is, based on the properties you will define.

Note: ECS ComposeX evaluates first for CacheCluster, so you might need to add an extra different parameter for ReplicationGroup to be detected appropriately.

26.2 MacroParameters

This allows you to define a very few of the AWS CacheCluster resource if you do not want to define the *Properties* and / or extra resources that are common to both the ReplicationGroup and CacheCluster.

Listing 2: Short syntax for properties to create a new CacheCluster

```
Engine: "redis|memcached"          # The engine, required.
EngineVersion: <engine_version>    # The engine version, required
CacheNodeType: <cache_node type>   # Optionally, define the CacheNodeType, defaults␣
↪to cache.t3.small
NumCacheNodes: <N>                 # Optionally, define the NumCacheNodes, defaults␣
↪to 1
ParameterGroup: {}                 # Optioanlly, define a new parameter group
```

26.2.1 ParameterGroup

This allows you to create a specific parameter group for the CacheCluster or ReplicationGroup. It supports all of the properties you can set in the original AWS ParameterGroup definition.

Hint: Your parameter group settings have to match the settings supported by the Engine. Refer to Engine Parameters guide to see what the engine you have can support as settings.

26.3 Settings

See *Settings*

26.4 Services

```
Services:
  - name: <service name>     # Service or Family name
    access: <ignored>        # Generic property that has to be set, ignored for now.
```

List of services you want to grant access to the CacheCluster or ReplicationGroup to. ECS ComposeX will automatically get the attributes of your cluster based on its type (Memcached/Redis/Redis ReplicationGroup), and pass these on down to the service stack.

Most importantly, it will create the SecurityGroup Ingress rules to allow your service to have access to the Cluster Node via the indicated SecurityGroup.

Hint: ECS ComposeX will not handle the Redis6.x RBAC access as this is a lot more involved than generating CFN templates etc. This might come in a future version.

26.5 Lookup

This allows you to define via Tags the ElasticCache Cluster or ReplicationGroup that already exists and you want your services to have access to.

It will automatically select the AWS Security Group associated with your cluster and put down the settings of your cluster into a CloudFormation mapping to pass it onto the services.

26.6 Examples

```
---
# ComposeX env file with ElasticCache definitions


x-elasticache:
  cache01:
    Properties:
```

(continues on next page)

(continued from previous page)

```
      AutoMinorVersionUpgrade: 'true'
      Engine: memcached
      EngineVersion: 1.6.6
      CacheNodeType: cache.t3.small
      NumCacheNodes: 1
    Services:
      - name: app03
        access: RW

  cache-02:
    MacroParameters:
      Engine: redis
      EngineVersion: 6.x
    Services:
      - name: app03
        access: RW

  cache03:
    Properties:
        ReplicationGroupDescription: my description
        NumCacheClusters: '2'
        Engine: redis
        CacheNodeType: cache.m3.medium
        AutoMinorVersionUpgrade: 'true'
        AutomaticFailoverEnabled: 'true'
        CacheSubnetGroupName: subnetgroup
        EngineVersion: 6.x
        PreferredMaintenanceWindow: 'wed:09:25-wed:22:30'
        SnapshotRetentionLimit: '4'
        SnapshotWindow: '03:30-05:30'
    Services:
      - name: app02
        access: RW
```

CHAPTER

TWENTYSEVEN

X-S3

27.1 Create or use existing S3 buckets to use for your applications

27.1.1 Properties

For the properties, go to to AWS CFN S3 Definition

27.1.2 MacroParameters

Some use-cases require special adjustments. This is what this section is for.

- *NameSeparator*
- *ExpandRegionToBucket*
- *ExpandAccountIdToBucket*

NameSeparator

Default is **-** which separates the different parts of the bucket that you might have automatically added via the other MacroParameters

As shown below, the separator between the bucket name and AWS::AccountId or AWS::Region is **-**. This parameter allows you to define something else.

Note: I would recommend not more than 2 characters separator.

Warning: The separator must allow for DNS compliance **[a-z0-9.-]**

ExpandRegionToBucket

When definining the *BucketName* in properties, if wanted to, for uniqueness or readability, you can append to that string the region id (which is DNS compliant) to the bucket name.

```yaml
Properties:
  BucketName: abcd-01
Settings:
  ExpandRegionToBucket: True
```

Results into

```yaml
!Sub abcd-01-${AWS::Region}
```

ExpandAccountIdToBucket

Similar to ExpandRegionToBucket, it will append the account ID (additional or instead of).

```yaml
Properties:
  BucketName: abcd-01
Settings:
  ExpandRegionToBucket: True
```

Results into

```yaml
!Sub 'abcd-01-${AWS::AccountId}'
```

Hint: If you set both ExpandAccountIdToBucket and ExpandRegionToBucket, you end up with

```yaml
!Sub 'abcd-01-${AWS::Region}-${AWS::AccountId}'
```

27.1.3 Services

As for all other resource types, you can define the type of access you want based to the S3 buckets. However, for buckets, this means distinguish the bucket and the objects resource.

Listing 1: permissions example

```yaml
x-s3:
  bucketA:
    Properties: {}
    Settings: {}
    Services:
      - name: service-01
        access:
          objects: RW
          bucket: ListOnly
```

27.1.4 Lookup

Lookup is currently implemented for S3 buckets!

Hint: For S3, if the S3 bucket has a default KMS key encryption, the services will automatically be granted KMS default **EncryptDecrypt** permissions in order to allow using the KMS key for objects manipulation.

27.1.5 IAM Permissions

For S3 buckets, the access types is expecting a object with **objects** and **bucket** to distinguish permissions for each. If you indicate a string, the default permissions (bucket: ListOnly and objects: RW) will be applied.

Listing 2: Full access types policies definitions

```
{
    "objects": {
        "CRUD": {
            "Action": [
                "s3:GetObject",
                "s3:DeleteObject",
                "s3:PutObject",
                "s3:GetObjectTagging",
                "s3:GetObjectVersionTagging",
                "s3:PutObjectTagging",
                "s3:PutObjectVersionTagging",
                "s3:DeleteObjectTagging",
                "s3:DeleteObjectVersionTagging",
                "s3:PutObjectAcl",
                "s3:AbortMultipartUpload",
                "s3:CreateMultipartUpload"
            ],
            "Effect": "Allow"
        },
        "RW": {
            "Action": [
                "s3:GetObject*",
                "s3:PutObject*"
            ],
            "Effect": "Allow"
        },
        "StrictRW": {
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Effect": "Allow"
        },
        "StrictRWDelete": {
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Effect": "Allow"
```

(continues on next page)

(continued from previous page)

```
        },
        "RWDelete": {
            "Action": [
                "s3:GetObject*",
                "s3:PutObject*",
                "s3:DeleteObject*"
            ],
            "Effect": "Allow"
        },
        "ReadOnly": {
            "Action": [
                "s3:GetObject*"
            ],
            "Effect": "Allow"
        },
        "StrictReadOnly": {
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow"
        },
        "WriteOnly": {
            "Action": [
                "s3:PutObject*"
            ],
            "Effect": "Allow"
        },
        "StrictWriteOnly": {
            "Action": [
                "s3:PutObject"
            ],
            "Effect": "Allow"
        }
    },
    "bucket": {
        "ListOnly": {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:GetBucketPublicAccessBlock"
            ]
        },
        "PowerUser": {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucket*",
                "s3:SetBucket*"
            ]
        }
    }
}
```

27.1.6 Examples

Listing 3: Create new S3 buckets

```
version: "3.8"

x-s3:
  bucket-01:
    Properties:
      BucketName: bucket-01
      AccessControl: BucketOwnerFullControl
      ObjectLockEnabled: True
      PublicAccessBlockConfiguration:
          BlockPublicAcls: True
          BlockPublicPolicy: True
          IgnorePublicAcls: True
          RestrictPublicBuckets: False
      AccelerateConfiguration:
        AccelerationStatus: Suspended
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: "aws:kms"
              KMSMasterKeyID: "aws/s3"
      VersioningConfiguration:
        Status: "Enabled"
    MacroParameters:
      ExpandRegionToBucket: True
      ExpandAccountIdToBucket: True
    Settings:
      EnvNames:
        - bucket01
        - BUCKET_ABCD-01
    Services:
      - name: app03
        access: RWObjects
  bucket-03:
    Properties:
      BucketName: bucket-03
      AccessControl: BucketOwnerFullControl
      ObjectLockEnabled: True
      PublicAccessBlockConfiguration:
          BlockPublicAcls: True
          BlockPublicPolicy: True
          IgnorePublicAcls: True
          RestrictPublicBuckets: False
      AccelerateConfiguration:
        AccelerationStatus: Suspended
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: AES256
      VersioningConfiguration:
        Status: "Enabled"

    Settings:
      ExpandRegionToBucket: True
```

(continues on next page)

(continued from previous page)

```yaml
      ExpandAccountIdToBucket: False
      EnvNames:
        - bucket01
        - BUCKET_ABCD-01
    Services:
      - name: app03
        access: RWObjects
  bucket-02:
    Properties: {}
    Settings:
      ExpandRegionToBucket: False
      ExpandAccountIdToBucket: False
      EnableEncryption: AES256
      EnableAcceleration: True
      EnvNames:
        - bucket01
        - BUCKET_ABCD-01
    Services:
      - name: app03
        access:
          bucket: ListOnly
          objects: RW

  bucket-04:
    Properties:
      BucketName: bucket-04
    Settings:
      NameSeparator: "."
      ExpandRegionToBucket: False
      ExpandAccountIdToBucket: False
      EnableEncryption: AES256
      EnableAcceleration: True
      EnvNames:
        - bucket01
        - BUCKET_ABCD-01
    Services:
      - name: app03
        access:
          bucket: ListOnly
          objects: RW
```

Listing 4: Lookup and use only existing buckets

```yaml
version: "3.8"

x-s3:
  bucket-07:
    Lookup:
      Tags:
        - aws:cloudformation:logical-id: ArtifactsBucket
        - aws:cloudformation:stack-name: pipeline-shared-buckets
    Services:
      - name: app03
        access:
          bucket: PowerUser
          objects: RW
```

(continues on next page)

(continued from previous page)

```
  bucket-08:
    Settings:
      EnvNames:
        - BUCKET03
    Lookup:
      Name: sacrificial-lamb
      Tags:
        - composex: "True"
    Services:
      - name: app03
        access:
          bucket: PowerUser
          objects: RW
```

Listing 5: Create new bucket with AWS CFN properties

```
version: "3.8"

x-s3:
  bucket-01:
    Properties:
      BucketName: bucket-01
      AccessControl: BucketOwnerFullControl
      AccelerateConfiguration:
        AccelerationStatus: Suspended
      ObjectLockEnabled: True
      PublicAccessBlockConfiguration:
        BlockPublicAcls: True
        BlockPublicPolicy: True
        IgnorePublicAcls: True
        RestrictPublicBuckets: False
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: "aws:kms"
              KMSMasterKeyID: "aws/s3"
      VersioningConfiguration:
        Status: "Enabled"
      MetricsConfigurations:
        - Id: EntireBucket
      LifecycleConfiguration:
        Rules:
          - Id: GlacierRule
            Prefix: glacier
            Status: Enabled
            ExpirationInDays: '365'
            Transitions:
              - TransitionInDays: '1'
                StorageClass: GLACIER
      CorsConfiguration:
        CorsRules:
          - AllowedHeaders:
              - '*'
            AllowedMethods:
              - GET
```

(continues on next page)

(continued from previous page)

```
      AllowedOrigins:
        - '*'
      ExposedHeaders:
        - Date
      Id: myCORSRuleId1
      MaxAge: '3600'
    - AllowedHeaders:
        - x-amz-*
      AllowedMethods:
        - DELETE
      AllowedOrigins:
        - 'http://www.example.com'
        - 'http://www.example.net'
      ExposedHeaders:
        - Connection
        - Server
        - Date
      Id: myCORSRuleId2
      MaxAge: '1800'
   WebsiteConfiguration:
    IndexDocument: index.html
    ErrorDocument: error.html
    RoutingRules:
     - RoutingRuleCondition:
         HttpErrorCodeReturnedEquals: '404'
         KeyPrefixEquals: out1/
       RedirectRule:
         HostName: ec2-11-22-333-44.compute-1.amazonaws.com
         ReplaceKeyPrefixWith: report-404/
   NotificationConfiguration:
    TopicConfigurations:
     - Topic: 'arn:aws:sns:us-east-1:123456789012:TestTopic'
       Event: 's3:ReducedRedundancyLostObject'
  MacroParameters:
   ExpandRegionToBucket: True
   ExpandAccountIdToBucket: True
  Settings:
   EnvNames:
    - bucket01
    - BUCKET_ABCD-01
  Services:
   - name: app03
    access: RWObjects
```

CHAPTER

TWENTYEIGHT

X-EFS

As described in the *volumes* documentation, in order to setup an AWS EFS Filesystem, you can either use the ECS Plugin definition, which will let ECS Compose-X import and define default settings, or alternatively, you can define your own settings using **x-efs**.

Attention: For more details around permissions and access to the filesystem, refer to *Filesystem, Access Point and services access*

28.1 Syntax reference

```
volumes:
  abcd:
    x-efs:
      Properties: {}
      MacroParameters: {}
      Settings: {}
      Lookup: {}
      Use: <str>
```

Hint: Even though x-efs is defined at the volumes level, at rendering time, a top level EFS stack will be created to contain the various filesystems required to be shared access across services.

28.2 Properties

As usual, the Properties supported as equal to the properties you would define in native CloudFormation. Refer to the AWS CFN EFS syntax reference for more details.

28.3 MacroParameters

However, AWS EFS has evolved since and some very tidy and neat features have emerged since, such as the EFS Access Points.

As it is ECS Compose-X objective to abstract that complexity away from developers but retain the security to high standards, we have implemented simple feature(s) to automatically enable using features such as IAM Authentication to further control access.

28.3.1 EnforceIamAuth

Listing 1: Enable IAM Auth restriction

```yaml
volumes:
  abcd:
    x-efs:
      MacroParameters:
        EnforceIamAuth: <True|False>
```

The purpose of IAM Authentication is to allow applications to authenticate against an EFS Access Point which will allow for further security configuration, such as, setting UID/GID to use, among others.

But primarily this will allow connection to the EFS using the Task IAM Role as a way to authenticate a specific application which can then translate into specific files access permissions.

When using IAM Authentication, this also enforces to use TLS between the client and the server, for increased security.

By enabling this feature, an access point will be created specifically for your services in the task definition, along with the filesystem.

Attention: To use that feature, it is highly recommend to use the EFS Mount Helper

28.4 Settings

This might be one rare case where the generic **EnvNames** has no impact, given that the volume name is the only thing that matters in this particular use-case. ECS Will automatically resolve the DNS name of the target in order to mount the shared filesystem as a volume to the container.

28.4.1 Subnets

As for other services that require to be created in a VPC to be accessed (for EFS, via Mount Targets), you can override the default behaviour (for EFS, defaults to the StorageSubnets).

28.5 Lookup

As usual, the Plug N' Play aspect of ECS Compose-X to your existing infrastructure is a key concern, therefore, you can also use ECS Compose-X to identify dynamically AWS EFS which already exists.

```yaml
volumes:
  abcd:
    x-efs:
      Lookup:
        Tags: []
        RoleArn: <>
```

28.6 Use

If you did know your Filesystem ID in AWS EFS, and wanted to just pass it on as the value instead of using Lookup, you can, either through use or through the original ECS Plugin definition.

Listing 2: ECS Plugin syntax

```yaml
volumes:
  abcd:
    external: true
    name: fs-abcd1234
```

Listing 3: ECS ComposeX Syntax

```yaml
volumes:
  abcd:
    x-efs:
      Use: fs-abcd1234
```

28.7 Examples

A full example using Bitnami Wordpress image (which requires users permissions etc. to be set) can be found in GitHub

28.8 Filesystem, Access Point and services access

AWS EFS has a notion of Access Point, which are very well described in the docs and other blog articles on the AWS sites. In a nutshell, they will allow you to control access to the Filesystem and "proxy" your access so that your services can set use specific POSIX users and root folders to the filesystem

This comes in to be very important if you are using a shared EFS among multiple tenants (applications, services etc.) and want to ensure separation for each but not have to spend hours configuring each service clients.

28.8.1 Access point per "container" within the task definition

In ECS Compose-X there is only so much that we can understand from the settings set at the volumes level. Given ECS Compose-X tries to focus as much as possible on security, we have implemented the following:

- If your task definition only has 1 container definition, there is one volume created in the task level, used by containers
- **If there is more than one container definition and you defined a different user property for the service, a new** access point is created specifically for that container, added to the task definition.

Warning: Even with 1 access point per container in the task definition, the access remains at the task level for IAM permissions.

CHAPTER

TWENTYNINE

X-APPMESH

- *Syntax*
- *Properties*
  - *MeshName*
  - *MeshOwner*
  - *EgressPolicy*
- *Settings*
  - *nodes*
    - *Syntax*
    - *Examples*
  - *routers*
    - *Definition*
    - *Syntax*
    - *match*
      - *Definition*
      - *Syntax*
      - *Example*
  - *services*
    - *Syntax*
  - *Examples*
- *AWS AppMesh & AWS Cloud Map for services mesh & discovery*
- *Nodes*
- *Routers*
  - *Services*
  - *The other things ECS ComposeX takes care of for you*

Warning: This module is still under development and we would love to get any feedback on the syntax and how to make it easier.

29.1 Syntax

```
x-appmesh:
  Properties:
    MeshName: str
    MeshOwner: str
    EgressPolicy: str
  Settings:
    Nodes:
      - <node>
    Routers:
      - <router>
    Services:
      - <service>
```

The properties for the mesh are very straight forward. Even though, the wish with ECS ComposeX is to keep the Properties the same as the ones defined in CFN as much as possible, for AWS AppMesh, given the simplicity of the properties, we are going with somewhat custom properties, mostly to allow for more features integration down the line.

Warning: There is only one mesh that will be either created or used to deploy the services into.

```
x-appmesh:
  Properties: {}
  Settings: {}
```

29.2 Properties

29.2.1 MeshName

This is the name of the mesh. However, if you do not specify the *MeshOwner*, then the name is ignored and the root stack name is used.

The MeshName is going to be used if you specify the MeshOwner, in case you are deploying into a Shared Mesh.

```
AllowedPattern: ^[a-zA-Z0-9+]+$
```

29.2.2 MeshOwner

The MeshOwner as described above, doesn't need to be specified, if you are creating your Nodes, Routers and Services (virtual ones) into a Mesh shared with you from another account.

```
AllowedPattern: [0-9]{12}
```

29.2.3 EgressPolicy

The mesh aims to allow services, nodes to communicate to each other only through the mesh. So by default, ECS ComposeX sets the policy to *DROP_ALL*. Meaning, no traffic out of the nodes will be allowed if not to a defined VirtualService in the mesh.

For troubleshooting and otherwise for your use-case, you might want to allow any traffic to get out of the node anyway. If so, simply change the policy to *ALLOW_ALL*

```
AllowedValues: DROP_ALL, ALLOW_ALL
```

29.3 Settings

The settings section is where we are going to define how our services defined in Docker compose are going to integrate to the mesh.

29.3.1 nodes

Syntax

```
Name: str # <family name>
Procotol str
Backends:
  - <service_name> # Only services can be defined as backend
```

Examples

This section represents the nodes. The nodes listed here must be either a service as listed in docker-compose or a family name.

```
Nodes:
  - Name: app01
    Procotol Http
  - Name: app02
    Procotol Tcp
    Backends:
      - service-abcd
```

29.3.2 routers

Definition

Routers as mentioned in the module description, are here to allow developers to define how packets should be routed from one place to another.

For TCP ones, one can only really set timeout settings, in addition to TLS etc. However for Http, Http2 and gRPC it allows you to define further more rules. The example below shows how a request to the router on path **/** it should send requests with the POST method to app02, but requests with the GET method to app01.

Syntax

```
Name: str
Listener
  Procotol str
  port: int
Routes:
  Http:
    - <match>
```

match

This is simplistic version of the AWS Route Match specifications : HTTP Route, TCP Route

Definition

The match allows to define how to route packets to backend nodes

Syntax

```
Match:
  Prefix: str
Method: str
Scheme:: str
Nodes:
  - <node_name>
```

Example

```
Routers:
  - Name: Httprouter
    Listener
      Procotol Http
      port: 8080
    Routes:
      Http:
        - Match:
            Prefix: /
```

(continues on next page)

(continued from previous page)

```
          Method: GET
          Scheme:: Http
          Nodes:
            - app01
        - Match:
            Prefix: /
          Method: POST
          Nodes:
            - app02
```

29.3.3 services

The VirtualServices are what acts as backends to nodes, and as receiver for nodes and routers. The Virtual Services can use either a Node or a Router as the location to route the traffic to.

Syntax

```
Services:
  - Node: <node_name>
    Name: str
  - Router: <router_name>
    Name: str
```

```
Services:
  - Name: service-xyz
    Router: Httprouter
  - Name: service-xyz
    Node: app03
```

29.3.4 Examples

```
---
# Simple mesh definition for new mesh of Services

x-appmesh:
  Properties: {}
  Settings:
    Nodes:
      - Name: app03
        Protocol: Tcp
      - Name: youtoo
        Protocol: Http
      - Name: bignicefamily
        Protocol: Http
        Backends:
          - dateteller # Points to the dateteller Service, not Router!
    Routers:
      - Name: dateteller
        Listener:
          Port: 5000
          Protocol: Http
```

(continues on next page)

(continued from previous page)

```
        Routes:
          Http:
            - Match:
                Prefix: /date
                Method: GET
                Scheme: Http
              Nodes:
                - Name: youtoo
                  Weight: 1
      - Name: datetellertcp
        Listener:
          Port: 5000
          Protocol: Tcp
        Routes:
          Tcp:
            - Nodes:
                - Name: app03
                  Weight: 1
    Services:
      - Name: api
        Node: bignicefamily
      - Name: dateteller
        Router: dateteller
```

29.4 AWS AppMesh & AWS Cloud Map for services mesh & discovery

AWS AppMesh is a service mesh which takes care of routing your services packets logically among the different nodes. What this allows you to do, it to explicitly declare which services have access to others, either on http, tcp or gRPC.

See also:

ComposeX *x-appmesh* syntax reference

Note: For HTTP, it supports both http2 and http.

There are a lot more features to know about, so I would recommend to head to the AWS Appmesh official documentation.

Warning: At the time of working on this feature, mutualTLS is not available, for lack of $$ to use AWS ACM CA and do the dev work.

Warning: By default in ECS ComposeX, the EGRESS policy for nodes it to DROP_ALL so that only explicitly allowed traffic can go across the mesh, in/out the services.

29.5 Nodes

The nodes are a logical construct to indicate an endpoint. With ECS ComposeX, it will either be

- a service defined and deployed in ECS
- a database
- any DNS discoverable target.

When you enable AWS AppMesh in ECS ComposeX, it will automatically add all the necessary resources for your ECS task to work correctly:

- envoy container
- update task definition with proxy configuration
- add IAM permissions for envoy to discover services and the mesh settings.

29.6 Routers

Routers are logical endpoints that apply the logic you define into routes. For TCP routers, it mostly is about defining TCP settings, such as timeouts.

For HTTP and gRPC however, it is far more advanced. You can define routes based on path, method etc. I also can perform healthcheck for you, to evaluate the nodes health. It effectively is a virtual ALB listener with a long set of rules.

Note: From experimenting and testing however, you cannot mix routes protocols within the same router.

29.6.1 Services

The virtual services are once again, a logical pointer to a resource. That resource will either be a Node or a Router. But again, it is aimed to be a virtual pointer, therefore, **you do not need to call your virtual service with the same name as one of the services defined in the compose services**.

What does that mean?

In essence, when you define a VirtualService as the backend of a virtual node, this means this node and its services will be granted access to the nodes of the VirtualService itself. But, you might have called your services **clock** and **watch**, and yet the virtual service will be called **time**.

Problem: when trying to connect to the endpoint **time**, your application won't be able to resolve **time**. Solution: ECS ComposeX will create a virtual service in the same AWS CloudMap as where the ECS Services are registered, and create a fake instance of it, for which the IPv4 address will be **169.254.255.254** How does it work?: your microservice in ECS will try to resolve **time**. The DNS response will be an IP address, here, **169.254.255.254**. Which obviously does not exist in a VPC (see RFC 3927 for more details) but, it will allow your application to establish the connection. The connection is intercepted by the envoy proxy container, which internally figures out, where to connect and how. It will then take your package, and send it across to the destination, **to the right IP address**. **Which is why resolving the IP in DNS is important, but the value of the record is not.**

29.6.2 The other things ECS ComposeX takes care of for you

In addition to configuring the ECS Task definition appropriately etc, ECS ComposeX also will take care of the security groups opening between the Virtual Nodes, and to other backends.

Yes, a mesh with DROP_ALL will ensure that communication between nodes only happens if explicitly allowed, but this does not mean we should not also keep the underlying network in check.

The security group inbound rule defined is from the source node to the target node(s), allowing all traffic *for now* between the nodes.

Note: For troubleshooting, you can use the ClusterWide Security Group which is attached to all containers deployed with ECS ComposeX, and allow all traffic within the security group to allow your ECS Services to communicate.

CHAPTER

THIRTY

X-DNS

Allows you to indicate what the DNS settings shall be for the deployment of your containers.

30.1 Syntax

Listing 1: Private Namespace definition (Uses AWS CloudMap)

```
PrivateNamespace:
  Name: str # TLD to use for the deployment.
  Lookup: str # Domain name to find in CloudMap
  Use: str # Expects the CloudMap ns- namespace ID
```

Warning: This domain will be associated with the VPC Route53 "database". If another Namespace using the same domain name already is associated with the VPC, this will fail.

Listing 2: Public DNS Zone using Route53.

```
PublicZone:
  Name: str # TLD to use for the deployment.
  Lookup: str # Domain name to find in CloudMap
  Use: str # Expects the CloudMap Z[A-Z0-9]+- Hosted Zone Id
```

Attention: For ACM DNS Validation and other validations to work, the zone must be able to be resolved.

30.2 Examples

Listing 3: Private definition only

```
x-dns:
  PrivateNamespace:
    Name: mycluster.lan
```

Listing 4: Public Zone and private zone

```yaml
x-dns:
  PrivateNamespace:
    Name: mycluster.lan
    Use: ns-abcd012344
  PublicZone:
    Name: public-domain.net
    Use: Z0123456ABCD
```

CHAPTER
THIRTYONE

X-ELBV2

This module allows you to define Application and Network Load-Balancers (Gateways not tested yet), and define which of your services should receive traffic, and add settings such as health check etc.

31.1 Syntax

```
x-elbv2:
  lbA:
    Properties: {}
    MacroParameters: {}
    Services: []
      - name: str
        protocol: str
        port: int
        healthcheck: str
    Listeners: []
```

31.2 Properties

For this particular resource, the only attributes that match the CFN definition that ECS Compose-X will import are

- Scheme
- Type
- LoadBalancerAttributes

All other settings are automatically generated for you based on the network and security definitions you have defined in the services and targets section.

Subnets associations can be overridden in the Settings.Subnets section. See *Subnets* for more details.

Hint: For Application Load Balancers, a new security group will be created automatically. Subnets are selected automatically based on the scheme you indicated. If selected a public NLB, the EIP addressed will automatically be provisioned too.

31.3 MacroParameters

Listing 1: ELBv2 Macro Parameters

```
timeout_seconds: int
desync_mitigation_mode: str
drop_invalid_header_fields: bool
http2: bool
cross_zone: bool
Ingress: {}
```

31.3.1 Ingress

Similar syntax as for ECS Services Ingress, allow you to define Ingress.

Tip: When using NLB, ingress must be defined at the service level, as NLB do not have a SecurityGroup

Listing 2: Ingress Syntax

```
Ingress:
  ExtSources: []
  AwsSources: []
```

Listing 3: ExtSources syntax

```
ExtSources:
  - Name: str (if any non alphanumeric character set, will be deleted)
    Description: str
    IPv4: str
```

Listing 4: AwsSources syntax

```
AwsSources:
  - Type: SecurityGroup|PrefixList (str)
    Id: sg-[a-z0-9]+|pl-[a-z0-9]+
    Lookup: {}
```

Tip: You can use either Id or Lookup to identify the SecurityGroups. Check out the *Lookup* syntax reference

31.3.2 Other attribute shortcuts

These settings are just a shorter notation for the LB Attributes

Shorthand	AttributeName	LB Type
timeout_seconds	idle_timeout.timeout_seconds	ALB
desync_mitigation_mode	routing.http.desync_mitigation_mode	ALB
drop_invalid_header_fields	routing.http.drop_invalid_header_fields.enabled	ALB
http2	routing.http2.enabled	ALB
cross_zone	load_balancing.cross_zone.enabled	NLB

31.4 Services

This follows the regular pattern of having the name of the service and access, only this time in a slightly different format. The services represent the Target Group definition of your service. Once again, in an attempt to keep things simple, you do not have to indicate all of the settings exactly as CFN does.

The Targets will automatically be pointing towards the ECS Service tasks.

31.4.1 name

Given that you can now re-use one of the service in the docker-compose file multiple times for multiple ECS Services in multiple Task definitions, and ECS to ELBv2 supports to route traffic to a specific container in the task definition, you have to indicate the service name in the following format

```
# name: <family_name>:<service_name>
name: youtoo:app01
name: app03:app03
```

Hint: If you service is not associated to a family via deploy labels, the family name is the same as the service name.

31.4.2 protocol

The Target Group protocol

31.4.3 port

The port of the target to send the traffic to

Hint: This port is the port used by the Target Group to send traffic to, which can be different to your healthcheck port.

31.4.4 healthcheck

The healthcheck properties can be defined in the same fashion as defined in the Target Group definition. However, it is also possible to shorten the syntax into a simple string

```
(port:protocol)(:healthy_count:unhealthy_count:intervals:timeout)?(:path:http_codes)?
```

Note: The last part, for path and HTTP codes, is only valid for ALB

31.5 Listeners

You can define in a very simple way your Listener definition and cross-reference other resources, here, the services and ACM certificates you might be creating.

It has its own set of properties, custom to ECS ComposeX.

The following properties are identical to the original CFN definition.

- Port
- Protocol
- SslPolicy
- Certificates

Hint: For certificates, you can also use **x-acm** to refer to an ACM certificate you are creating with this stack. It will automatically import the Certificate ARN and map it once created.

Hint: You can re-use the same ACM certificate defined in x-acm for multiple listeners. Make sure to have all the Alt. Subjects you need!

Warning: The certificate ARN must be valid when set, however, we are not checking that it actually exists.(yet)

31.6 Target Groups

List of targets to send the requests to. These are equivalent to ELBv2::TargetGroup

```yaml
name: <service_name> ie. app03:app03
access: <domain name and or path> ie. domain.net/path
cognito_auth: AuthenticateCognitoConfig
```

This represents the targets and simultaneously the Listener Rules to apply so that you can point to multiple services at once and implement these rules.

31.6.1 name

The name of the family and service in that family to send the requests to.

31.6.2 access

Allows you to define the conditions based on the path or domain name (or combination of both) that should be in place to forward requests.

If you only define the domain name, any path in that domain will be what's matched.

31.6.3 AuthenticateCognitoConfig

Defines the AuthenticateCognitoConfig requirement condition / action

31.6.4 AuthenticateOidcConfig

Similar to AuthenticateCognitoConfig but for OIDC providers. This allows to respect all the AuthenticateOidcConfig Properties as per CFN definition.

Tip: We highly recommend that you store the OIDC details into a secret in secrets manager!

Hint: For both AuthenticateCognitoConfig and AuthenticateOidcConfig, the rules defined in *access* will be set to come **after** the authenticate action.

31.7 Examples

```
---
# ELBv2 creation for services


x-dns:
  PublicZone:
    Name: lambda-my-aws.io
    Use: ZABCDEFGHIS0123

x-acm:
  public-acm-01:
    Properties:
      DomainName: test.lambda-my-aws.io
      DomainValidationOptions:
        - HostedZoneId: ZABCDEFGHIS0123
          DomainName: test.lambda-my-aws.io
      SubjectAlternativeNames:
        - anothertest.lambda-my-aws.io
        - yet.another.test.lambda-my-aws.io
      ValidationMethod: DNS

x-elbv2:
  lbA:
    Properties:
      Type: application
    MacroParameters:
```

(continues on next page)

(continued from previous page)

```
      S3Logs: bucket:/prefix
      timeout_seconds: 60
      desync_mitigation_mode: defensive
      drop_invalid_header_fields: True
      http2: False
      cross_zone: True
      Ingress:
        ExtSources:
          - Ipv4: "0.0.0.0/0"
            Description: ANY
          - Ipv4: "1.1.1.1/32"
            Description: CLOUDFLARE
            Name: CLOUDFLARE
    Listeners:
      - Port: 80
        Protocol: HTTP
        DefaultActions:
          - Redirect: HTTP_TO_HTTPS
      - Port: 443
        Protocol: HTTP
        Certificates:
          - x-acm: public-acm-01
        Targets:
          - name: bignicefamily:app01
            access: /somewhere
      - Port: 8080
        Protocol: HTTP
        Certificates:
          - x-acm: public-acm-01
          - CertificateArn: arn:aws:acm:eu-west-1:012345678912:certificate/102402a1-
↪d0d2-46ff-b26b-33008f072ee8
        Targets:
          - name: bignicefamily:rproxy
            access: /
          - name: youtoo:rproxy
            access: /stupid
          - name: bignicefamily:app01
            access: thereisnospoon.ews-network.net:8080/abcd/test.html

    Services:
      - name: bignicefamily:rproxy
        port: 80
        healthcheck: 5000:HTTP:/healthcheck:200,201
      - name: bignicefamily:app01
        port: 5000
        healthcheck: 5000:HTTP:/path/to/healthcheck:200,201
      - name: youtoo:rproxy
        port: 80
        healthcheck: 5000:HTTP:5:2:15:3:/ping.This.Method:200,201

  lbC:
    Properties:
      Scheme: internet-facing
      Type: network
    MacroParameters:
      cross_zone: True
    Settings: {}
```

(continues on next page)

(continued from previous page)

```
    Listeners:
      - Port: 8080
        Protocol: TCP
        Targets:
          - name: app03:app03
            access: /
      - Port: 8081
        Protocol: TCP
        Certificates:
          - x-acm: public-acm-01
        Targets:
          - name: app03:app03
            access: /
    Services:
      - name: app03:app03
        port: 5000
        healthcheck: 5000:TCP:7:2:15:5
        protocol: TCP
```

Listing 5: ELBv2 with

```
x-elbv2:
  authLb:
    Properties:
      Scheme: internet-facing
      Type: application
    Settings: {}
    Listeners:
      - Port: 8080
        Protocol: HTTP
        Targets:
          - name: app03:app03
            access: /
      - Port: 8081
        Protocol: HTTP
        Targets:
          - name: app03:app03
            access: /
            AuthenticateOidcConfig:
              Issuer: "{{resolve:secretsmanager:/oidc/azuread/
↪app001:SecretString:Issuer}}"
              AuthorizationEndpoint: "{{resolve:secretsmanager:/oidc/azuread/
↪app001:SecretString:AuthorizationEndpoint}}"
              TokenEndpoint: "{{resolve:secretsmanager:/oidc/azuread/
↪app001:SecretString:TokenEndpoint}}"
              UserInfoEndpoint: "{{resolve:secretsmanager:/oidc/azuread/
↪app001:SecretString:UserInfoEndpoint}}"
              ClientId: "{{resolve:secretsmanager:/oidc/azuread/
↪app001:SecretString:ClientId}}"
              ClientSecret: "{{resolve:secretsmanager:/oidc/azuread/
↪app001:SecretString:ClientSecret}}"
              SessionCookieName: "my-cookie"
              SessionTimeout: 3600
              Scope: "email"
              AuthenticationRequestExtraParams":
                display": "page"
```

(continues on next page)

(continued from previous page)

```
                prompt": "login"
              OnUnauthenticatedRequest: "deny"
    Services:
      - name: app03:app03
        port: 5000
        healthcheck: 5000:HTTP:7:2:15:5
        protocol: HTTP
```

CHAPTER

THIRTYTWO

X-ACM

This module to allow people to create ACM certificates, auto-validate these with their DNS registration, and front their applications with HTTPS.

Hint: Recently got supported by CloudFormation to natively add the CNAME entry to your Route53 DNS record as the certificate is created, removing the manual validation process.

32.1 Syntax

```
x-acm:
  certificate-01:
    Properties: {} # AWS CFN Properties
    MacroParameters: {} # ComposeX Macro parameters for ACM
```

Warning: You cannot be creating your public DNS Zone and validating it at the same time, simply because the NS servers of you new Public Zone are not registered in your DNS registra. Therefore, DNS validation would never work. Make sure that if you are creating a new DNS PublicZone, you will be able to use it!

32.2 Properties

The properties will be supported exactly like in the native AWS ACM Properties

Hint: If you defined multiple **SubjectAlternativeNames** names, they will be auto-added to the validation list and use the same ZoneId, so you do not need to list them all in DomainValidationOptions

32.3 MacroParameters

In the aspiration of making things easy, you can now simply define very straight forward settings to define your certificate. This automatically creates the full ACM Certificate definition, and uses DNS validation.

```
DomainNames:
  - domain.tld
  - sub.domain.tld
HostedZoneId: ZoneID
```

32.3.1 DomainNames

List of the domain names you want to create the ACM Certificate for.

Hint: The first domain name will be used for the CN, and the following ones will be used for SubjectAlternative names

---

32.3.2 HostedZoneId

If you wish to override the x-dns/PublicZone settings you can set that here.

Note: That HostedZone ID will be used for *all* of the Domain Validation.

32.4 Services

No need to indicate services to assign the ACM certificate to. Refer to *x-elbv2* for mapping to ALB/NLB.

32.5 Example

```
x-acm:
  public-acm-01:
    Properties:
      DomainName: test.lambda-my-aws.io
      DomainValidationOptions:
        - HostedZoneId: ZABCDEFGHIS0123
          DomainName: test.lambda-my-aws.io
      SubjectAlternativeNames:
        - anothertest.lambda-my-aws.io
        - yet.another.test.lambda-my-aws.io
      ValidationMethod: DNS
```

Hint: If you need to specify *x-dns* in the template and provide the **HostedZoneId** which will be used there. DNS Reference: *x-dns*

CHAPTER

THIRTYTHREE

X-KINESIS

This module helps you create new Kinesis Data Streams supporting all the AWS CFN properties and link these streams to your services.

33.1 Syntax reference

Listing 1: x-kinesis Syntax reference

```
x-kinesis:
  stream:
    Properties: {} # AWS Kinesis CFN definition
    Settings: {}
    MacroParameters: {}
    Services: []
```

33.2 Properties

The Properties are the AWS CFN definition for AWS Kinesis streams.

33.3 MacroParameters

No specific MacroParameters for Kinesis data streams. Given the AWS definition is very straightforward, just define the properties. The only truly required property is the ShardCount

33.4 Settings

The settings are as usual, allow you to define *EnvNames*

33.4.1 EnvNames

List of String that allow you to define multiple environment names for the stream to be exposed to your service. Value for these is the **AWS Kinesis Stream name** (Default value returned by Fn::Ref

33.5 Services

As per the generic Services definition, we have a list of object, name and access, which define how the service can access the stream.

For AWS Kinesis streams, we have the following permissions.

- Producer
- Consumer
- PowerUser

33.6 Examples

Listing 2: Services definition example

```yaml
services: [serviceA, serviceB]

x-kinesis:
  streamA:
    Properties:
      ShardCount: 2
    Services:
      - name: serviceA
        access: Producer
      - name: serviceB
        access: Consumer
```

33.7 IAM permissions

Listing 3: IAM permissions pre-defined for your services.

```json
{
    "Consumer": {
        "Effect": "Allow",
        "Action": [
            "kinesis:Get*",
            "kinesis:DescribeStreamSummary"
        ]
    },
    "Producer": {
        "Effect": "Allow",
        "Action": [
            "kinesis:PutRecord"
        ]
    },
```

(continues on next page)

(continued from previous page)

```
    "PowerUser": {
        "Effect": "Allow",
        "NotAction": [
            "kinesis:CreateStream",
            "kinesis:DeleteStream"
        ]
    }
}
```

CHAPTER THIRTYFOUR

X-SQS

34.1 Define your AWS SQS Queues and service scaling based on messages queue depth

34.1.1 Syntax

Listing 1: SNS Syntax Reference

```
x-sns:
  QueueA:
    Properties: {}
    Settings: {}
    Services: []
```

34.1.2 Properties

Mandatory Properties

SQS does not require any properties to be set in order to create the queue. No settings are mandatory.

Special properties

It is possible to define Dead Letter Queues for SQS messages (DLQ). It is possible to easily define this in ECS ComposeX simply by referring to the name of the queue, deployed in this same deployment.

Warning: It won't be possible to import a queue ARN at this time in ECS ComposeX that exists outside of the stack today.

34.1.3 Services

Similar to all other modules, we have a list of dictionaries, with two keys of interest:

- name: the name of the service as defined in services
- access: the type of access to the resource.
- scaling: Allow to define the scaling behaviour of the service based on SQS Approximate Messages Visible.

IAM Permissions

- RO - read only
- RWMessages - read/write messages on the queue
- RWPermissions - read/write messages and grants access to modify some queue attributes

Tip: IAM policies, are defined in sqs/sqs_perms.json

Hint: You can also use AWS SAM Permissions as defined in AWS Documentation

Listing 2: SAM Policy Example

```yaml
services:
  serviceA: {}
x-sqs:
  QueueA:
    Services:
      - name: serviceA
        access: SQSPollerPolicy
```

34.1.4 Lookup

See *Lookup* for more details about Lookup.

```yaml
x-sqs:
  QueueA:
    Lookup:
      Tags:
        - Name: queue-a-123
        - owner: app01
```

34.1.5 Scaling

You can now defined StepScaling on the ECS Service based on the number of messages in the queue!

Listing 3: Scaling Syntax

```yaml
scaling:
  steps:
    - lower_bound: int
      upper_bound: int
      count: int
  scaling_in_cooldown: int
  scaling_out_cooldown: int
```

Tip: You can define scaling rules on SQS Queues that you are importing via *Lookup*

Attention: If you already setup other Scaling policies for the service, beware of race conditions!

34.1.6 Special Features

Redrive policy

The redrive policy works exactly as you would expect it and is defined in the exact same way as for within the SQS proprties. Only, here, you only need to put the queue name of the DLQ. The generated ARN etc. will be fetched via exports (which also implicitly adds a lock on it).

Example with DLQ:

```yaml
x-sqs:
  DLQ:
    Properties: {}
    Settings: {}
    Services: []

AppQueue:
  Properties:
    RedrivePolicy:
      deadLetterTargetArn: DLQ
      maxReceiveCount: 10
  Settings:
    EnvNames:
      - APPQUEUE01
```

34.1.7 Settings

Refer to *Settings*

34.1.8 Examples

Listing 4: Simple SQS Queues with DLQ configured

```
x-sqs:
  Queue02:
    Services:
      - name: app02
        access: RWPermissions
      - name: app03
        access: RO
    Properties:
      RedrivePolicy:
        deadLetterTargetArn: Queue01
        maxReceiveCount: 10
    Settings:
      EnvNames:
        - APP_QUEUE
        - AppQueue

  Queue01:
    Services:
      - name: app03
        access: RWMessages
    Properties: {}
    Settings:
      EnvNames:
        - DLQ
        - dlq
```

Listing 5: SQS Queue with scaling definition

```
x-sqs:
  QueueA:
    Services:
      - name: abcd
        access: RWMessages
        scaling:
          ScaleInCooldown: 120
          ScaleOutCooldown: 60
          steps:
            - lower_bound: 0
              upper_bound: 10
              count: 1 # Gives you 1 container if there is between 0 and 10 messages␣
↪in the queue.
            - lower_bound: 10
              upper_bound: 100
              count: 10 # Gives you 10 containers if you have between 10 and 100␣
↪messages in the queue.
            - lower_bound: 100
              count: 20 # Gives you 20 containers if there is 100+ messages in the␣
↪queue
```

Note: The last step cannot have defined a upper_bound. If you set one, it will be automatically be removed.

Note: You need to have defined x-configs/scaling/Range to enable step scaling on the ECS Service.

CHAPTER

THIRTYFIVE

X-SNS

35.1 Syntax

Listing 1: x-sns syntax reference

```
x-sns:
  Topics:
    TopicA:
      Properties: {}
      Settings: {}
      Services: []
  Subscriptions:
    SubscriptionA:
      Properties: {}
      Settings: {}
      Topics: []
```

Warning: At this current version, **Subscriptions** are not supported.

35.2 Properties

Refer to AWS SNS Topic Documentation for SNS Topics

35.3 Lookup

Lookup is currently implemented for SNS topics!

35.4 Examples

Listing 2: Create new topics

```
x-sns:
  Topics:
    abcd:
      Properties: {}
      Services:
        - name: app01
          access: Publish
        - name: you-too
          access: Publish
```

Listing 3: Create and Lookup SNS topics

```
x-sns:
  Topics:
    abcd:
      Properties: {}
      Services:
        - name: app01
          access: Publish
        - name: you-too
          access: Publish

    hello:
      Lookup:
        Tags:
          - costcentre: lambda
          - composexdev: "yes"
      Services:
        - name: app03
          access: Publish
```

CHAPTER

THIRTYSIX

X-EVENTS

This extension allows you to define an AWS EventBride rule to stop start services at specific times of the day or based on specific events.

36.1 Properties

You can find all the properties on the AWS CFN Events Rules definitions.

Note: You do not need to define Targets to point to the services defined in docker-compose. Refer to *Services* for that.

36.2 MacroParameters

No specific parameters at this time!

36.3 Settings

No specific settings at this time!

36.4 Services

There we define the tasks we want to deploy at specific times or events.

Listing 1: Services syntax for rules

```
name: service_name
TaskCount: <N>
DeleteDefaultService: True/False (default. False)
```

36.4.1 name

Here we want to define the name of the **family** we want to use for trigger. If the service is not defined as part of a specific family, you can use the service name itself.

See also:

Required: Yes.

36.4.2 TaskCount

Same property as for ECS Parameters of the Task Rule target definition itself, this allows you to set a specific number of tasks.

Required: Yes.

Hint: Not using deploy/replicas on purpose, because of the *DeleteDefaultService* option

---

36.4.3 DeleteDefaultService

Custom setting, this allows you to NOT define a ECS Service along with the task, therefore you will only get the TaskDefinition created.

CHAPTER

THIRTYSEVEN

X-KMS

37.1 Syntax

```
x-kms:
  keyA:
    Properties: {}
    Settings: {}
    Services: []
    Lookup: {}
```

37.2 Properties

See AWS CFN KMS Key Documentation

37.3 Settings

37.3.1 Alias

In addition to EnvNames, for KMS, we also have **Alias** which will create an Alias along with the KMS Key. The alias name must be a string, not starting with alias/aws or aws. If you specify a an alias starting with **alias/** then the string will be used as is, if you only specify a short name, then the alias will be prefixed with the RootStack name and region.

37.4 Examples

Listing 1: Simple key creation and link to services

```
x-kms:
  keyA:
    Properties:
      PendingWindowInDays: 14
    Services:
      - name: serviceA
        access: EncryptDecrypt
      - name: serviceB
        access: EncryptDecrypt
```

(continues on next page)

(continued from previous page)

```
    Settings:
      Alias: keyA
```

37.5 Services

List of key/pair values, as for other ECS ComposeX x-resources.

Three access types have been created for the table:

- EncryptDecrypt
- EncryptOnly
- DecryptOnly
- SQS

Listing 2: KMS and Services

```
x-kms:
  keyA:
    Properties: {}
    Services:
      - name: serviceA
        access: EncryptDecrypt
      - name: serviceB
        access: DecryptOnly
```

37.6 IAM Permissions

Three access types have been created for the table:

- EncryptDecrypt
- EncryptOnly
- DecryptOnly
- SQS

Listing 3: KMS Permissions scaffold

```
{
    "SQS": {
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "Effect": "Allow"
    },
    "DecryptOnly": {
        "Action": [
            "kms:Decrypt"
        ],
        "Effect": "Allow"
```

(continues on next page)

(continued from previous page)

```
    },
    "EncryptOnly": {
        "Action": [
            "kms:Encrypt",
            "kms:GenerateDataKey*",
            "kms:ReEncrypt*"
        ],
        "Effect": "Allow"
    },
    "EncryptDecrypt": {
        "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:CreateGrant",
            "kms:DescribeKey"
        ],
        "Effect": "Allow"
    }
}
```

CHAPTER

THIRTYEIGHT

X-VPC

38.1 Define a new VPC for your services or use an existing one

38.1.1 Syntax Reference

```
x-vpc:
  Create: {}
  Lookup: {}
  Use: {}
```

Create

Listing 1: Create example with a single NAT and 3 VPC Endpoints

```
x-vpc:
  Create:
    SingleNat: true
    VpcCidr: 172.6.7.42/24
    Endpoints:
      AwsServices:
        - service: s3
        - service: ecr.api
        - service: ecr.dkr
```

VpcCidr

The CIDR you want to use. Default is **100.127.254.0/24**.

SingleNat

Whether you want to have 1 NAT per AZ for your application subnets. Reduces the costs for dev environments!

Endpoints

List of VPC Endpoints from AWS Services you want to create. Default will create Endpoints for ECR (DKR and API).

EnableFlowLogs

Whether you want to have a VPC Flow Log created for the VPC. It will create a new LogGroup and IAM Role to allow logging to CloudWatch.

FlowLogsRoleBoundary

For those of you who require IAM PermissionsBoundary for your IAM Roles, this allows to set the boundary. If it starts with **arn:aws** it will assume this is a valid ARN, otherwise, it will use the value as policy name.

38.1.2 Lookup

```yaml
x-vpc:
  Lookup:
    VpcId:
      Tags:
        - key: value
    PublicSubnets:
      Tags:
        - vpc::usage: public
    AppSubnets:
      Tags:
        - vpc::usage: application
    StorageSubnets:
      Tags:
        - vpc::usage: storage0
```

Warning: When using **Use** or **Lookup** you MUST define all 4 settings: * VpcId * StorageSubnets * AppSubnets * PublicSubnets

Warning: When creating newly defined subnets groups, the name must be in the format **^[a-zA-Z0-9]+$**

Hint: You can define extra subnet groups based on different tags and map them to your services for override when using **Lookup** or **Use**

Listing 2: Extra subnets definition

```yaml
x-vpc:
  Lookup:
    VpcId: {}
    AppSubnets: {}
    StorageSubnets: {}
```

(continues on next page)

(continued from previous page)

```yaml
    PublicSubnets: {}
    Custom01:
      Tags: {}

networks:
  custom01:
    x-vpc: Custom01


services:
  serviceA:
    networks:
      - custom01
```

38.1.3 Use

```yaml
x-vpc:
  Use:
    VpcId: vpc-id
    AppSubnets:
      - subnet-id
      - subnet-id
    StorageSubnets:
      - subnet-id
      - subnet-id
    PublicSubnets:
      - subnet-id
      - subnet-id
```

38.1.4 Default VPC Network design

The design of the VPC generated is very simple 3-tiers:

- Public subnets, 1/4 of the available IPs of the VPC CIDR Range
- Storage subnets, 1/4 of the available IPs of the VPC CIDR Range
- Application subnets, 1/2 of the available IPs of the VPC CIDR Range

Default range

The default CIDR range for the VPC is **100.127.254.0/24** This leaves a just under 120 IP address for the EC2 hosts and/or Docker containers.

Hint: The range can be changed via **VpcCidr** but not the structure detailed above. Works for all RFC 1918 and the 100.64.0.0/10 ranges.

CHAPTER THIRTYNINE

X-CLUSTER

This section allows you to define how you would like the ECS Cluster to be configured. It also allows you to define *Lookup* to use an existing ECS Cluster.

39.1 Properties

Refer to the AWS CFN reference for ECS Cluster

Listing 1: Override default settings

```
x-cluster:
  Properties:
    CapacityProviders:
      - FARGATE
      - FARGATE_SPOT
    ClusterName: spotalltheway
    DefaultCapacityProviderStrategy:
      - CapacityProvider: FARGATE_SPOT
        Weight: 4
        Base: 2
      - CapacityProvider: FARGATE
        Weight: 1
```

39.2 Lookup

Allows you to enter the name of an existing ECS Cluster that you want to deploy your services to.

Listing 2: Lookup existing cluster example.

```
x-cluster:
  Lookup:
    Tags:
      - name: clusterabcd
      - costcentre: lambda
```

Warning: If the cluster name is not found, by default, a new cluster will be created with the default settings.

39.3 Use

This key allows you to set a cluster to use, that you do not wish to lookup, you just know the name you want to use. (Useful for multi-account where you can’t lookup cross-account).

CHAPTER
FORTY

X-ALARMS

Listing 1: Syntax reference

```yaml
x-alarms:
  alarm-01:
    Properties: {}
    MacroParameters: {}
    Settings: {}
    Services: []
    Topics: []
```

40.1 Properties

ECS Compose-X will automatically detect whether your properties define an Alarm or a Composite Alarm.

See AWS CW Alarms definition and AWS CW Composite Alarms definition

> **Attention:** When linking to Services and/or Topics, the OKActions, AlarmActions will be overridden accordingly.

> **Attention:** You can only create new alarms. To use existing alarms with new services would required to modify the actions of that alarm, which would be an external change to any IaC.

40.2 MacroParameters

For x-alarms, MacroParameters is here to help define in a simpler way a composite alarm. More specifically, all you have to define is the Alarm expression

```yaml
MacroParameters:
  CompositeExpression: <str>
```

40.2.1 CompositeExpression

String with a logical high level expression of the composite alarm.

Hint: In your expression, use the alarm name as defined in the compose file, not using the **AlarmName** property! ECS Compose-X will automatically map to the CFN Alarm being created.

40.3 Services

```
x-alarms:
  kafka-scaling-01:
    Properties: {}
    Services:
      - name: <str>
        access: <str>
        scaling: {} # Service scaling definition
```

40.4 Topics

Listing 2: Topics syntax

```
x-alarms:
  alarms-01:
    Properties: {}
    Topics:
      - TopicArn: <str>
        NotifyOn: okay
      - x-sns: <str>
        NotifyOn: all
```

40.4.1 TopicArn

A string representing the topic ARN. The topic ARN must be valid (will be validated).

40.4.2 x-sns

Allows you to define a SNS topic that was defined in compose-x files already. Supports new created topics and topics found via Lookup.

40.4.3 NotifyOn

This allows you to determine whether the messages should be published based on the alarm status.

Value	Alarm actions
all	OKActions AlarmActions
alarm	AlarmActions
okay	OKActions

40.5 Examples

Listing 3: Alarm with scaling actions for service

```yaml
---
# x-alarms basic use-case

x-alarms:
  alarm-01:
    Properties:
      ActionsEnabled: True
      AlarmDescription: A simple CW alarm
      ComparisonOperator: GreaterThanOrEqualToThreshold
      DatapointsToAlarm: 1
      Dimensions:
        - Name: Cluster
          Value: DEV
        - Name: Topic
          Value: sainsburys.data.price-specification.batch.v1
        - Name: ConsumerGroup
          Value: sainsburys.applications.sc-dis.price-specification.retail-price.aut-
↪test-consumer
      EvaluationPeriods: 5
      MetricName: TotalLagForTopicAndConsumerGroup
      Namespace: lag-metrics-v4
      Period: 60
      Statistic: Sum
      Threshold: 1.0
      TreatMissingData: notBreaching


    Services:
      - name: app03
        access: NA
        Scaling:
          scaling_in_cooldown: 300
          scaling_out_cooldown: 60
          steps:
            - lower_bound: 0
              upper_bound: 1000
              count: 1
            - lower_bound: 1000
              upper_bound: 10000
              count: 3
```

(continues on next page)

(continued from previous page)

```
    Topics:
      - TopicArn: arn:aws:sns:eu-west-1:012346578900:topic/sometopic
      - x-sns: topic-01


x-sns:
  Topics:
    topic-01:
      Properties: {}
```

Listing 4: Example CompositeAlarm with MacroParameters

```
x-alarms:
  alarm-01:
    Properties {}

  alarm-02:
    Properties: {}

  composite-alarm:
    MacroParameters:
      CompositeExpression: ALARM(alarm-01) and ALARM(alarm-02)
```

Hint: When the alarms is only for the service, the alarm gets created in the same stack as the service(s). When the alarm has both services and topics, the alarms are created in a separate stack.

Hint: When defining a composite alarm, the actions defined in *Services* or *Topics* are ignored.

CHAPTER FORTYONE

SPOT_CONFIG

This module is not strictly a module which the same settings as the other AWS resources. This is a module which allows users to create the EC2 compute resources necessary to run the ECS Containers on top of EC2 workloads.

Note: At this point in time, there is no support for creating Capacity providers in CloudFormation, therefore we cannot implement that functionality.

Note: By default, everything is built to use EC2 spot fleet, simply to save money on deployment for testing. Future will allow to run pure OnDemand or hybrid mode.

41.1 Define settings in the configs section

At the moment, the settings you can change for the compute definition of your EC2 resources are defined in

configs -> globals -> spot_config

Example:

```yaml
x-configs:
  spot_config:
    bid_price: 0.42
    use_spot: true
    spot_instance_types:
    m5a.xlarge:
      weight: 4
    m5a.2xlarge:
      weight: 8
    m5a.4xlarge:
      weight: 16
```

With the given AZs of your region, it will create automatically all the overrides to use the spot instances.

Note: This spotfleet comes with a set of predefined Scaling policies, in order to further reduce cost or allow for scaling out based on EC2 metrics.

Warning: We cannot recommend any more to use AWS Fargate and configure your capacity providers instead of EC2 instances. Use with caution

CHAPTER

FORTYTWO

DOCKER ECS PLUGIN SUPPORT

Soon after the Open source release of the Compose definition, AWS and Docker worked on a new docker plugin, the *ecs-plugin* which allows to perform some similar tasks as with ECS ComposeX.

However, these fields usually will require full ARN of your resources, whereas ECS ComposeX will allow you to do discovery of your resources and I hope give you a lot more flexibility.

With that said, the objective of ECS ComposeX is to help developers and so I added the support for the ECS Plugin extensions fields.

See also:

Docker and ECS official documentation

42.1 x-aws-cluster

As per the official documentation, this allows you to define the ARN of an ECS Cluster you have that you want to use to deploy the services into.

If left empty, a new cluster gets created.

With ComposeX you can use the expected ARN to indicate which cluster to deploy to. Equally, you can provide just the name of the Cluster, ComposeX will filter it out of the ARN and behave in a similar fashion as **x-cluster/Use**

See also:

x-cluster

42.2 x-aws-pull_credentials

This allows you to define the secret in secrets manager that contains the username/password for authentication with a private docker image registry.

With ComposeX you can either use it as is defined in the official documentation or combine it with the docker-compose secrets.

Listing 1: Example of ARN use

```
services:
  app01:
    image: private.registry.mydomain.net/repository-app01
    x-aws-pull_credentials: "arn:aws:secretsmanager:eu-west-1:012345678912:secret:/
↪path/to-creds"
```

Listing 2: Example with docker-compose secret definition

```yaml
secrets:
  private_repository:
    x-secrets:
      Name: /path/to/creds

services:
  app02:
    image: private.registry.mydomain.net/repository-app02
    x-aws-pull_credentials: secrets::private_repository
```

Hint: For either methods, this will add the RepositoryCredentials property to the Task definition and add an IAM policy to the Execution Role to *secretsmanager:GetSecretValue*

Hint: When using the ECS ComposeX way, you can use all the existing features of secrets (Lookup etc).

Warning: You cannot use JsonKeys for this secret.

42.3 x-aws-autoscaling

This setting allows you to define autoscaling configuration for your service. With the ECS Plugin you can define CPU and RAM autoscaling which are assigned to the ECS Service.

If in your docker-compose files you have not defined **x-scaling** this will be used to define the scaling policies.

However, in case you set both **x-aws-autoscaling** and **x-scaling**, the latter will be used and the x-aws-autoscaling settings are ignored.

This is by design as **x-scaling** allows for a lot more settings to be defined than **x-aws-autoscaling**

42.4 x-aws-policies

This allows to define additional IAM policies that are assigned to the ECS Task Role. It behaves exactly in the same way as **x-iam/ManagedPolicies** does.

Listing 3: ECS Plugin syntax

```yaml
services:
  foo:
    x-aws-policies:
      - "arn:aws:iam::aws:policy/AmazonS3FullAccess"
```

Listing 4: ECS Compose-X syntax

```yaml
services:
  foo:
```

(continues on next page)

(continued from previous page)

```
    x-iam:
      ManagedPolicies:
        - "arn:aws:iam::aws:policy/AmazonS3FullAccess"
```

42.5 x-aws-role

Allows to defined extra IAM policies. However, not that the ECS Plugin is going to automatically generate the name of the policy assigned to the ECS Task Role.

ECS ComposeX syntax is a little lengthier to get to the IAM policies. However, allows you to define your own policy and you can have multiple ones.

Listing 5: ECS Plugin syntax

```
services:
  foo:
    x-aws-role:
      Version: "2012-10-17"
      Statement:
        - Effect: "Allow"
          Action:
            - "some_aws_service"
          Resource:
            - "*"
```

Listing 6: ECS ComposeX Syntax

```
services:
  foo:
    x-iam:
      Policies:
        - PolicyName: SomeName
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: "Allow"
                Action:
                  - "some_aws_service"
                Resource:
                  - "*"
```

Hint: For x-aws-role and x-aws-policies, ECS ComposeX will not override what you had defined and instead simply merge the two definitions.

Hint: If you need to defined IAM permissions boundary, you can with ECS Compose-X. *x-iam*

42.6 x-aws-logs_retention

Allows you to define the CloudWatch Log Group RetentionInDays period. When used in combination with ComposeX **x-logging**, the highest(max) value will be used as we consider you might want the longest period for tracking purposes.

If either is set and the other is not, the value is set accordingly.

Listing 7: Example with just x-aws-logs_retention

```yaml
services:
  serviceA:
    x-aws-logs_retention: 42
```

Listing 8: Both x-logging and x-aws-logs_retentions defined. Here, 64 will be set.

```yaml
services:
  serviceA:
    x-logging:
      RetentionInDays: 42
    x-aws-logs_retention: 64
```

See also:

x-logging

Hint: If you set an arbitrary value that would not be a valid value for AWS logs retention, ComposeX will automatically match to the closest valid value. For example, for 42, this will be 30. For 64, it will be 60.

42.6.1 x-aws-min_percent & x-aws-max_percent

This allows to define the percentages for ECS Deployment Configuration.

```yaml
services:
  serviceA:
    x-aws-min_percent: 50
    x-aws-max_percent: 150
    deploy:
      replicas: 4
      update_config:
        failure_action: rollback
```

CHAPTER FORTYTHREE

HISTORY

43.1 0.14.0 (2021-03-23)

Version 0.14.0 is a release coming with a new LICENSE attached, the Mozilla Public License 2.0 (MPL 2.0).

- 1e82eed LICENSE change to MPL-2.0 (John Preston)

43.1.1 New features

- 9fbe3aa New pre-defined alarms for services (#432) (John Preston)
- a6083d7 Added CompositeAlarm support (#431) (John Preston)

43.1.2 Fixes

- 534dcd0 reversed conditions logic for IAM Role for SAR template (John Preston)
- 9f145cf Publish template for AWS SAR (#438) (John Preston)
- 8008043 Removing the scaling target and scaling policies (#436) (John Preston)
- 122efae Fixed output attribute name for S3 to RDS feature (#433) (John Preston)

43.1.3 Improvements

- 1eeb6f6 Upgrade to Troposphere 2.7.0 (John Preston)
- 2afec02 Improved macro settings override and layer key (#440) (John Preston)
- 51a568f new cfn-macro Parameter BucketName (#439) (John Preston)
- ef08ae9 New image URL for XRay (John Preston)
- 670bf27 Adding default prefix for default log group name (#428) (John Preston)

43.2 0.13.0 (2021-03-10)

This new version comes with a good mix of fixes and new features supported. In an effort of always improving docker-compose compatibility, a number of features have been added. Volumes support is added for both local volumes (non-bind) and shared volumes (via EFS). Alarm support added to allow creating arbitrary alarms and scaling policies on metrics for non Compose-X managed resources.

43.2.1 New Features

- 33f7b45 x-alarms support (#425)
- e12d25a ECS DeploymentConfiguration support with Circuit breaker (#423)
- dad6d02 awslogs drivers options support (#422)
- b66876b Added lookup for SecurityGroups in Ingress (#401)
- c3c1565 x-efs (#395)
- df7d085 Added tmpfs support
- d19e60d Added sysctls support
- 8c4c30e Added working_dir support
- 71cb736 Added shm_size support
- a09d233 Added cap_add,cap_drop support
- 69bc348 Added support for Ulimits
- 3f380c7 docker-compose ECS local volumes support (#391)

43.2.2 Fixes

- 811f88d Fixing URLs
- cae1336 build can be either a string or dict
- f093931 Fixed self-ingress process (#417)
- ec3dbc4 Fixing VpcId.Use and x-dns when not set (#415)
- f0d6635 Fixing lookup resource output condition (#411)
- 6dbef07 Fixing s3 to ecs bug for lookup (#400)
- 7edc838 Renamed and fixed condition for registries (#392)
- 8876047 For PrivateNamespace in CloudMap, using ns-ID (#388)
- b7130ea Family name is as defined in compose files, and LB use that name instead of logical name (#386)

43.2.3 Improvements

- 765426b Updated docs
- 07c6db2 Using troposphere 2.6.4
- 7a31e63 Simpler regexp to group required, ping and optional healthcheck (#416)
- 4977767 x-elbv2 settings in macro parameters for LB Attributes (#410)
- 0ea035a Code Cleanup and Refactor (#409)
- 8059454 Moved x-s3 settings to MacroParameters and cleaned up old unused code (#407)
- 8773299 Healthcheck times translated from str to int (#406)
- 5a49890 When not public NLB, allows to override the LB Subnets to use (#402)
- 695624f Added compatibility matrix (#398)
- ec184fc Generic attributes output configuration (#396)
- 5f1cc0b Adding a message to inform that no port were defined but UseCloudmap (#387)

43.3 0.12.0 (2021-01-31)

43.3.1 New features

- dd9246c Allowing to define features by names and related resources (#376) John Preston
- 2d0ef6d Allow to define RoleArn for DNS Lookup (#377) John Preston
- d85fd90 Add an IAM Role to RDS for S3import feature (#373) John Preston

43.3.2 Fixes

- b690d60 Fixing ingress parsing for Ingress (#382) John Preston
- 01c0582 Fix import value for subnets to Join for custom subnets (#381) John Preston
- 8f2b777 Passing the subnets as a string with !Join from mappings (#380) John Preston
- d72e9c1 Fixed events. Dumbed down the Fargate version John Preston
- 913d451 Fixing AppMesh
- 397c4cf Fixed ACM certificate mapping (#366) John Preston
- f09ad64 Fix S3 name generation, events subnet param (#357) (jacku7) Jack Saunders

43.3.3 Improvements

- 95f76ab Updated lookup based to be more accurate (#378) John Preston
- 62b27f7 Documentation updates/fixes and macro install/usage guide (#372) John Preston
- 1e77c87 Working lookup of DNS zones. Relies on DNS Name only. John Preston
- 5a8b659 VPC and subnets now in mappings John Preston
- 913d451 Zones require name John Preston
- 54593eb ECS Cluster "pointer" as a variable of settings John Preston
- d801463 * Files pulled for remote files are stored with tempfile * Fixing x-dns John Preston
- 0267cbc Refactor of DNS into more gracious handling John Preston
- e56b667 * Refactored ECS Cluster creation for simplicity John Preston
- ba511dd Create a nightly manifest list pointing always to the latest (#364) John Preston
- 3596286 Docker image release-work (#363) John Preston
- 02591ce Support for OIDC and Cognito AUTH action in x-elbv2 (#339) John Preston
- fb36420 Updating build conditions and methods (#362) John Preston
- 06d5776 Adding sitemap and meta keywords (#360) John Preston
- 29e75ef Re-arranging test files and patching up CI files (#361) John Preston

43.3.4 Special changes

The following changes all relate to the release a CFN Macro of ECS Compose-X

- 1aea413 Allow to set override Function IAM Role John Preston
- b804360 Maintain policy on previous layer versions (#383) John Preston
- 5fe8169 Adding retain policy on layer version permissions (#374) John Preston
- ae3d42a AWS Lambda Layer build and release (#371) John Preston
- 2b1c21b Adding macro image build phase and deploy template (#370) John Preston

43.4 0.11.0 (2021-01-14)

First release of 2021 focusing on some new features / extension of existing features, as well on improving stability.

43.4.1 New features

885e89e - DB Secrets exposable to services (#356) (John Preston) b723cc7 - Allow to override subnets to use for resources deployed inside VPC (#353) (John Preston) 0c6c86c - Create PrefixList for VPC and suibnets when creating a new VPC (#352) (John Preston) 4405fef - Support for ElasticCache Cluster via x-elasticache (#350) (John Preston) 59ceae0 - Added support for CodeGuru Profiling Group (#323) (John Preston) 97529fa - x-docdb support for DBClusterParameterGroup (#349) (John Preston) a8888b6 - Extending ecs-plugin x-fields support (#336) (John Preston)

43.4.2 Improvements

faed0d3 - Align to CamelCase for x-scaling and x-network settings (#347) (John Preston) 249ba18 - Moved defauls into properties dicts. Added more docstrings for clarity (#345) (John Preston) 97345c7 - Pyup/updates (#329) (John Preston) 774640b - Create pyup.io config file (#327) (pyup.io bot)

43.4.3 Fixes

8d14ac0 - Fix for use_cloudmap (#346) (John Preston) aa1ba40 - Fixed properties update (#344) (John Preston) d2cd544 - Fixing VPC related settings (#341) (John Preston)

43.5 0.10.0 (2020-12-13)

43.5.1 New features

- 976e5bb Support for env_file (#318)
- a432763 Import simple SAM IAM policies templates. (#316)
- db2c8fe Support for service-to-service explicit ingress (#300)
- fe1e0af Added to support DB Snapshot for new DB creation (#297)
- 73cdf9a x-vpc - Support for VPC FlowLogs (#296)
- b9f1ec8 Scaling rules for Lookup queues (#293)
- 54faa50 Feature x-dns::Records to add Public DNS Records pointing to elbv2 (#289)
- d5a97a1 Adding support for kinesis streams (#287)

43.5.2 Improvements

- 1be3b99 Improved secrets JsonKeys based on suggestions (#322)
- 6302bc6 x-rds:: Refactor Properties/MacroParameters/Settings (#309)

43.5.3 Fixes

- 191d420 No interpolate ${AWS::PseudoParameters} (#324)
- de87457 Bug fixes for RDS/DocDB and ECS containers (#305)
- 4220d7d TMP solution pending AWS official XRay publish (#304)
- 2c1fcfc Fix/duplicate secrets keys (#303)
- 4befc25 Fixed backward logic (#301)

43.5.4 Other updates and corrections

- 31d7bcc Added kinesis docs (#313)
- 997f0d9 Added back exports but not using in ComposeX. For cross-stacks usage (#310)
- cb0be55 Linted up code (#307)
- 5e559f0 Prefixing the log group with the root stack name for uniqueness (#295)
- c81f443 Refactored to single function recursively evaluating properties (#291)
- 16a5d39 Code linting (#285)

43.6 0.9.0 (2020-11-26)

43.6.1 New features

- cabd793 - Support for networks: and mapping to additional subnets. (#282)
- ba4ed5c - ECS Scheduled tasks support (#280)
- 82e2086 - Defaulting to encrypted for RDS (#276)
- a516a09 - Added support for service level x-aws keys from ecs-plugin (#273)
- 5e1ab08 - Improved logging settings (#265)
- 96ad398 - x-secrets::Lookup (#256)
- dfb249c - Lookup for ACM working (#254)
- ea6e05c - Feature x-docdb (#252)
- 0a4d258 - Refactor services to root stack (#248)
- 49a9d31 - ARN of TGT Group always passed to service stack (#245)
- eafcd38 - Updated documentation (#236)
- aa4c96b - Feature x-elbv2 with x-acm support and validation via x-dns (#228)
- fb0bc4a - Allowing RoleArn in x-rds Lookup (#233)
- 22feb56 - Lookup via resources tag api for VPC resources (#231)
- be536c1 - Cross-Cccount assume role generally and locally for lookup (#229)
- 32075f2 - Allow for custom cooldown for steps (#221)
- ca89836 - Upgrading troposphere==2.6.3 (#216)
- 3a1b0c8 - Linting DynDB features and use-case files (#213)
- 67cc67e - Feature x-s3 (#196)
- 230a9d3 - Lookup RDS DB/Clusters and secrets (#211)

43.6.2 Fixes

- fc55f4b - Patched version of 0.8.9 with previews for 0.9.0 (#275)
- 1dc4113 - Replaced LOG.warn with LOG.warning (#271)
- 42c7027 - Docs improvements (#278)
- 78bef91 - Clarified Ingress syntax (#261)
- af31f33 - Fixed a number of small issues (#259)
- 02da4e1 - Hotfix services attributes (#243)
- fb7265a - During PyCharm refactor, error change occured (#238)
- c46c208 - Fixing import export string (#224)
- 7669799 - Removing missed print (#217)
- 4171044 - Fixing condition when QueueName property is set (#210)
- 0ced643 - Patched SQS based scaling rule and alarm (#202)

43.6.3 Syntax changes from previous version

- 86d2141 - Refactor/services xconfig keys (#269)
- 1cfa6b7 - Refactor AppMesh properties keys (#262)
- d753473 - Refactor to classes for XResources and Compose resources (#219)

Documentation theme changed to Read The Docs and tuned some colors.

43.7 0.8.0 (2020-10-09)

43.7.1 New features:

- Support for ECS Scaling based on SQS Messages in queue
- Support for ECS Scaling based on Service CPU/RAM values (TargetTracking)
- Support for using existing Secrets in AWS Secrets Manager
- Support for Service logs expiry from compose definition
- Enable to use AWS CFN native PseudoParameters in string values
- Improved Environment variables interpolation to follow the docker-compose behaviour

43.7.2 Closed reported issues:

- https://github.com/compose-x/ecs_composex/issues/175

Some code refactor and bug fixes have gone in as well to improve stability and addition of new services.

43.8 0.7.0 (2020-08-12)

New features:

- Support for AWS Secrets mapping to secrets in docker-compose
- Support for *Use* on VPC which needs no lookup
- Support for IAM policies to manually add ad-hoc permissions outside of the pre-defined ones
- Additional configuration file to use with CodePipeline

Various bug fixes and some small features to help making plug-and-play easier. Introduction to *Use* which should allow for resources reference outside of your account without cross-account lookup.

43.9 0.6.0 (2020-08-03)

New features: * Docker-compose multi-files (override support)

The new CLI uses positional arguments matching a specific command which drives what's executed onwards. Trying to re-implement features as close to the docker-compose CLI as possible.

- **config** allows to get the YAML file render of the docker-compose files put together.
- **render** will put all input files together and generate the CFN templates accordingly.
- **up** will deploy do the same as render, and deploy to AWS CFN.

43.10 0.5.3 (2020-07-30)

A lot of minor bug fixes and removing CLI commands to the benefit of better implementation via the compose file.

43.11 0.5.2 (2020-07-30)

New features:

- Support for AWS KMS

The support for KMS will be extended to use the CMK for RDS/SQS/SNS and any resource that can use KMS for encryption at rest.

Hint: Mind, this might occur a few extra costs.

43.12 0.5.1 (2020-07-28)

Small bug patches and code refactoring. SQS now into a single stack unless there are more than 30 queues.

43.13 0.5.0 (2020-07-27)

43.13.1 New features

- DynOAamoDB support
- Lookup for existing tables which the services get IAM access to.

43.14 0.4.0 (2020-07-20)

- ACM Support for ALB/NLB for public services.
- AWS AppMesh support
- Attempt to making navigation through docs better.
- Automatic release to https://nightly.docs.ecs-composex.lambda-my-aws.io/ from master

To help with code quality and support, I subscribed to the following services:

- CodeScanning using SonarCloud.io
- CodeCoverage reports with Codecov

43.15 0.3.0 (2020-06-21)

Refactored the way the services, task definitions and containers are put together, in order to support multiple new features:

- Allow multiple services to be merged into one Task definition
- Support Docker compose v3 compute definition

The support for Docker compose compute settings allows to add up all the CPU / RAM of your service(s) and identify the closest Fargate CPU/RAM configuration for the **Task Definition** (the respective CPU/RAM of each task is unchanged).

The docker-compose file is now more strictly close to the definition set in Docker Compose, with regards to attributes and their expected types.

Note: In order to respect more closely the docker-compose definition, the key previously used **configs** now is **x-configs**

43.16 0.2.3 (2020-04-16)

Refactored the ecs part into a class and reworked the configuration settings to allow for easier integration. Documentation has been updated to reflect the changes in the structure of the configs section.

43.16.1 New features

- **Enable AWS X-Ray (#56)** Enabling X-Ray will allow developer to get APM metrics and visualize the application interaction with other services.
- **No-upload (#64)** This allows to store the templates locally only.

  Note: The templates are still validated from their body

- **IAM Boundary for the IAM roles (#55)** Permissions boundary are an IAM feature that allows to set boundaries which superseed other permissions associated to the entity. It is often the put as a condition for users creating roles to assign a specific Permission Boundary policy to the roles created.

43.17 0.2.2 (2020-04-10)

Refactor of the ECS service template into a single class (still got to be reworked). Refactored the ECS Services into a master class which ingests the CLI kwargs directly.

Reworked and reorganized documentation to help with readability

43.18 0.2.1 (2020-05-03)

Code refactored to allow a better way to go over each template and stack so everything is treated in memory before being put into a file and uploaded into S3.

- **Issues closed**
  - Docs update and first go at IAM perms (#22)
  - Refactor of XModules logic onto ECS services (#39)
  - Templates & Stacks refactor (#38)
  - Update issue templates for easy PRs and Bug reports
  - Added *make conform* to run black against the code to standardize syntax (#26)
  - Allow to specify directory to write all the templates to in addition to S3. (#27)
  - Reformatted with black (#25)
  - Expand TagsSpecifications with x-tags (#24)
  - Bug fix for root template and Cluster reference (#20)

Documentation structure and content updated to help navigate through modules in an easier way. Documented syntax reference for each module

43.18.1 New features

- **#6 - Implement x-rds. Allows to create RDS databases with very little properties needed**
    - Creates Aurora cluster and DB Instance
    - Creates the DB Parameter Group by importing default settings.
    - Creates a common subnet group for all DBs to run into (goes to Storage subnets when using –create-vpc).
    - Creates DB username and password in AWS SecretsManager
    - Applies IAM permissions to ECS Execution Role to get access to the secret
    - Applies ECS Container Secrets to the containers to provide them with the secret values through Environment variables.

43.19 0.1.3 (2020-04-13)

A patch release with a lot of little features added driven by the writing up of the blog to make it easier to have in a CICD pipeline.

See overall progress on GH Project

43.19.1 Issues closed

- Issue 14
- Issue 15

43.20 0.1.2 (2020-04-04)

Patch release aiming to improve the CLI and integration of the Compute layer so that the compute resources creation in EC2 are standalone and can be created separately if one so wished to reuse.

43.20.1 Issues closed

Issue related to the fix.

PR related to the fix.

43.21 0.1.1 (2020-04-02)

Added tags definition from Docker ComposeX with the x-tags which allows to add tags to all resources that support tagging from AWS CFN

```
x-tags:
  - name: TagA
    value: SomeValue
  - name: CostcCentre
```

(continues on next page)

(continued from previous page)

```
    value: IamNotPayingForThis
  - name: Some:Special:Key
    value: A long weird value
```

or alternatively in an object/dict format

```
x-tags:
  TagA: ValueA
  TagB: ValueB
```

43.22 0.1.0 (2020-03-24)

- **First release on PyPI.**
    - Working VPC + Cluster + Services
    - Working expansion of existing Cluster with new VPC
    - Working expansion of existing VPC and Cluster with new services
    - IAM working to allow services access to SQS queues
    - SQS Queues functional with DLQ
    - Works on Python 3.6, 3.7, 3.8
    - Working start of build integration in CodeBuild for automated testing

CHAPTER

FORTYFOUR

EXTRAS

44.1 Plug & Play to existing resources with auto discovery

Since the start of this project, the ability to plug & play to an existing infrastructure has been a priority to this project.

At the very beginning of it, it was mostly based out of finding one specific ID for a specific resource and it was rather complex and somewhat prone to errors, as software always is.

It has evolved since and improved significantly to allow a more flexible approach, relying nearly only on your resources and their tags.

Resource tags are a fantastic way to identify and distinguish resources from one another. More often than not, the resource ID of a resource, will be better left generated by AWS so you can later update it further without requiring replacement. But then one often finds a friendly tag, **Name** which allows us to know at a glance what the resource is about.

Now, looking for resources in your own account is easy but sometimes, you might need to be able to cross borders and identify shared resources into another AWS account.

You will find in the Lookup feature an option to specify a specific IAM role to use in order to perform the API calls. This IAM role corresponds to the IAM role in your "other" account which will give you permissions to look around for your resources.

Finally, alternatively, if you do not have cross-account in place or have some resources untagged, you can simply use the **Use** feature and provide directly the ID or ARN (depends based on the resource type) that you wish to use.

For further information, refer to *Lookup*

44.2 Docker ECS-Plugin x-aws-keys support

In order to keep make the integration and inter-operability of tools used by developers, we are going to add support for, mostly, services level x-aws keys such as **-xaws-iam-role** or **x-aws-autoscaling**.

This will allow developers who might have started a journey to ECS using the docker ecs plugin to continue that journey with ECS Compose-X without making too many changes.

In case for a similar setting, such as *x-aws-iam-policies* which in ECS Compose-X is under *x-iam/Policies*, these non conflicting settings will add up together. However, in case of conflicting information, the ECS Compose-X definition will prevail over the x-aws-keys.

44.3 AWS AppMesh integration

AWS AppMesh is a service mesh which allows you to define how services talk to each other at an application L7) level, and optionally, TCP (layer 4) level. It is extremely powerful

Since the beginning of the project, we have been using AWS Cloud Map to create a private DNS Hosted Zone linked to the VPC created at the same time. This allowed us to very simply register into the PHZ (private hosted zone) via Service Discovery.

We are going to use these entries to make a 1-1 mapping between our services defined in the *services* section of the docker-compose file and the *nodes* listed in the *x-appmesh* section.

AppMesh uses envoy as a side-car proxy that will capture our services packets and route these to their defined backends. Using AWS AppMesh empowers developers to declare how services are supposed to communicate together, what to do in case of errors, and administrators can define whether or not the traffic between all the components should be done using TLS termination end-to-end, to ensure no man-in-the-middle attacks could happen.

The syntax for AppMesh in ECS Compose-X is a mix of Istio, Envoy and AWS AppMesh definitions.

See also:

x-appmesh

44.4 Services autoscaling integration

You can now define scaling for your ECS Services using * CPU / RAM Target Tracking scaling * SQS Messages (visible) depth with step scaling.

For example, we want to scale our front-end based on CPU usage and our backend, dealing with queues, based on messages numbers.

```
services:
  frontend:
    ports:
      - 80:80
    image: my-nginx
    deploy:
      replicas: 2 # by default I want 2 containers
    x-scaling:
      Range: "1-10" # 1 to 10 containers to deploy for the service
        TargetScaling:
          CpuTarget: 80 # Means 80% average for all containers in the service.
  backend:
    image: my-worker
    deploy:
      replicas: 1  # Initially I want 1 container running to make sure everything is␣
↪working
    x-configs:
      scaling:
        Range: "0-10" # I can have between 0 to 10 containers. 0 because I am happy␣
↪not paying when nothing to do

x-sqs:
  jobs-queue:
    Properties: {}
    Settings: {}
    Services:
```

(continues on next page)

(continued from previous page)

```
      - name: frontend
        access: RWMessages
      - name: backend
        access: RWMessages
        scaling:
          steps:
            - lower_bound: 0
              upper_bound: 10
              count: 1
            - lower_bound: 10
              upper_bound: 20
              count: 2
            - lower_bound: 20
              count: 21
```

As you can see we defined scaling for SQS only on the backend, as we don't need to scale the frontend based on that. Also we set the count for final step to 21, which is higher than the Range indicated.

Our frontend will be managed by ECS itself which will be ensuring that the average CPU usage across the service remains under 80%.

Hint: In composex, you must define a generic Range first, and if you override it in the scaling, it will take the highest count of all scaling policies.

Note: Scaling with target tracking based on ELBv2 metrics is coming too.

44.5 Fargate CPU/RAM auto configuration

When you want to create services on ECS, you first need to create a Task Definition. Among the IAM permissions and the network configuration, the Task definition also defines how much CPU and RAM you want to have available **for all your containers** in the task.

If you have only one service, you might as well just not put any limits at the **Container Definition** level, and let it use all the available CPU and RAM defined in the **Task Definition**.

Hint: The Task definition CPU and RAM is the maximum CPU and RAM that your containers will be able to use. The amount of CPU and RAM in AWS Fargate is what determines how much you are paying.

But when you start to add side-cars, such as Envoy, X-Ray, or your WAF, your reverse-proxy, you want to start setting how much CPU and RAM these containers can use out of the Task Definition.

In docker-compose (or with swarm), you already have the ability to define the CPU limits and reservations you want to give to each individual service in the compose file.

To help having to know the different CPU/RAM settings supported by AWS Fargate, ECS Compose-X, if defined, will automatically use the limits and reservations configuration set in your Docker compose file, and determine what is the closest CPU/RAM configuration that will allow your services to run into.

Hint: Setting at least the reservation values so your containers are guaranteed some capacity in case other containers get to use more resources than expected.

See also:

deploy reference.

We have the following example:

```yaml
---
# Blog applications base file for testing

version: '3.8'
services:
  rproxy:
    image: ${IMAGE:-nginx}
    ports:
      - 80:80
    deploy:
      replicas: 2
      resources:
        reservations:
          cpus: "0.1"
          memory: "32M"
        limits:
          cpus: "0.25"
          memory: "64M"
    depends_on:
      - app01

  app01:
    image: ${IMAGE:-nginx}
    ports:
      - 5001
    deploy:
      resources:
        reservations:
          cpus: "0.25"
          memory: "64M"
    environment:
      LOGLEVEL: DEBUG
      SHELLY: ${SHELL}
      TERMY: "$TERM"
    links:
      - app03:dateteller

  app02:
    image: ${IMAGE:-nginx}
    ports:
      - 5000
    deploy:
      resources:
        reservations:
          cpus: "0.25"
          memory: "64M"
    environment:
      LOGLEVEL: DEBUG

  app03:
```

(continues on next page)

(continued from previous page)

```
    image: ${IMAGE:-nginx}
    ports:
      - 5000
    deploy:
      resources:
        reservations:
          cpus: "0.25"
          memory: "64M"
    environment:
      LOGLEVEL: DEBUG
    volumes:
      - shared-images:/shared/images
    secrets:
      - abcd
      - zyx


volumes:
  shared-images: {}

secrets:
  zyx:
    external: True
```

We have CPU and RAM limits set for **both limits and reservations**. So we know that we can use the limits, add them up, and this will indicate us our CPU configuration.

Hint: In docker compose, you indicate the CPU as a portion of vCPU. A value of 1.0 means 1024 cycles, or 1vCPU. A value of 0.25 equals to 256 cycles, which equivals to .25 of a vCPU.

We get: * 0.75 vCPU (limits) * 192MB of RAM.

The closest configuration for Fargate that will cater for the amount of vCPU, is 1024. With 512 only, we **could** run low in cpu cycles.

So then, from there, we know that Fargate will allow for a minimum of 2GB of RAM. So our CPU/RAM configuration will be **1024 CPU cycles and 2048MB of RAM**.

Now, let's say we know that our rproxy (NGINX based) will only need .1 CPU at most and 128M of RAM, and we want to make sure that the application container, does not take all the CPU and RAM away from it, but also that it should not go over these limits.

So we are going to set these limits for the rproxy container.

Hint: If you do not set the reservations, the container could potentially free compute resources to the benefit of others, but at the risk of having none available.

Now, let's say we know our application will use a minimum of 256M, and up to .25 of a CPU.

Let's count: * .1 vCPU (limit+reservation) and .25 (reservation). We get 0.35vCPU. * 128MB RAM (limit+reservation) and 256M (reservation), We get 284MB.

The closest configuration for Fargate is .5vCPU and 1024MG of RAM. But, also, our application container can use up to 1024-128 = 896MB of RAM, as we did not set a limit. For some applications where you are not totally sure of the RAM you might need, this is a good way to keep for free space, just in case.

Note: Chances are, if you are using so low CPU/RAM for your microservice, you might be running it in AWS Lambda!

Hint: You might think that for the CPU you need, ie. 1vCPU, which means you need at least 2GB of RAM for the appropriate Fargate profile, is a lot of RAM wasted.

However, in this configuration, the CPU represents ~80% of the costs (29.5$+6.5$=36$).

44.6 Multiple services, one microservice

Hint: Refer to *labels* for more details.

Regularly developers will build locally multiple services which are aimed to work together as a group. And sometimes, these services have such low latency requirements and dependency on each other, that they are best executed together.

In our example before, where we use NGINX to implement webserver logic, configuration and security, and leverage the power of a purpose-built software, as opposed to re-implement all that logic directly in your application, we might to run these two together.

On your workstation, when you run *docker-compose up*, it obviously is going to run it all locally. However, by default, these are defined as individual services.

To allow multiple services to be merged into a single **Task Definition**, and still treat your docker images separately, you can use a specific label that **ECS Compose-X** will recognize to group services into what we called a **family**.

ECS already has a notion of *family*, so I thought, we should use that naming to group services logically.

The deploy labels are ignored on a container level, therefore, none of these tags will show when you deploy the services.

Hint: The labels can be either a list of strings, or a "document" (dictionary).

But then you might wonder, how come are the permissions going to work for the services?

Remember, the permissions are set at the **Task definition** level. So any container within that service, will get the same permissions.

However, for the database as an example, which creates a Secret in AWS Secrets Manager, which we would then expose to the service with the *Secrets* attribute of the **Container Definition**, ECS Compose-X will specifically add that secret to that container only. Equally, for the services linked to SQS queues or SNS topics (etc.), the environment variable providing with the ARN of the resource, will also only expose the value to the container set specifically.

In case you wanted to allow an entire *family* of services to get access to the resources, you can also give, as the service name in the definition, the name of one of your families defined via the labels.

For example,

```yaml
services:
  worker01:
    image: worker01
    deploy:
```

(continues on next page)

(continued from previous page)

```
      labels:
        ecs.task.family: app01

  worker02:
    image: worker02
    deploy:
      labels:
        ecs.task.family: app01

x-sqs:
  Queue01:
    Properties: {}
    Services:
      - name: app01
        access: RWMessages
```

CHAPTER

FORTYFIVE

PHILOSOPHY

CloudFormation is awesome, the documentation is excellent and the format easy. So ECS Compose-X wants to keep the format of resources Properties as close to the orignal as possible as well as making it easier as well, just alike resources like **AWS::Serverless::Function** which will create all the resources around your Lambda Function as well as the function.

45.1 Trying to implement DevOps starting with developers

Whilst this is something that can be used by AWS Cloud Engineers tomorrow to deploy applications on ECS on the behalf of their developers, the purpose of ECS Compose-X is to enable developers with a simplistic and familiar syntax that takes away the need to be an AWS Expert. If tomorrow developers using Compose-X feel comfortable to deploy services by themselves, I would be able to stop hand-holding them all the time and focus on other areas.

45.2 Community focused

Any new Feature Request submitted by someone other than myself will get their request prioritized to try address their use-cases as quickly as possible.

Submit your Feature Request here

45.3 Ensure things work

It takes an insane amount of time to test everything as, generating CFN templates is easy, testing that everything works end-to-end is a completely different thing.

I will always do my best to ensure that any new feature is tested end-to-end, but shall anything slip through the cracks, please feel free to report your errors here

45.4 Provision other AWS resources your services need

So you have the definitions of your services and they are running on ECS. But what about these other services that you need for your application to work? DBs, notifications, streams etc. Are you going to run your MySQL server onto ECS too or are you going to want to use AWS RDS? How are you going to define the IAM roles and policies for each service? Access Secrets? Configuration settings?

That is the second focus of ECS Compose-X: defining extra sections in the YAML document of your docker compose file, you can define, for your databases, queues, secrets etc.

ECS Compose-X will parse every single one of these components. These components can exist on their own but what is of interest is to allow the services to access these.

That is where ECS Compose-X will automatically take care of all of that for you.

For services like SQS or SNS, it will create the IAM policies and assign the permissions to your ECS Task Role so the service gets access to these via IAM and STS. Credentials will be available through the metadata endpoint, which your SDK will pick immediately.

For services such as RDS or ElasticCache, it will create the security groups ingress rules as needed, and when applicable, will handle to generate secrets and expose these via ECS Secrets to your services.

45.5 How does it work?

To do so, ECS Compose-X will use the library called Troposphere and generate all the CloudFormation templates for it. These extra resources that you need (RDS, SQS etc.), need to be defined. To keep things simple, you can defined them in the same way you would do in AWS CloudFormation templates, add these resources to your compose definition.

Hint: x- is ignored by docker-compose when you run it. See Extensions fields

Note: x- and y- are natively defined in the YAML Specifications

CHAPTER

FORTYSIX

WHAT DOES ECS COMPOSE-X DO DIFFERENTLY? LONG VERSION

Where ECS Compose-X distinguishes itself from other tools is embedding security for each service individually, so that developers only have to connect resources logically together in the same way they would use links between microservices in their Docker Compose definition.

Each microservice needs to explicitly be declared as a consumer of a resource to get access to it, otherwise it won't be able to access the resource or other microservices.

This is achieved simply by using AWS IAM policies or security groups ingress, where applicable.

That simplified way to define access between services and resources helps with defining a shared-responsibility model between application engineers and cloud engineers:

Application engineers must know what their application does and how services interface to each other and to external services. This gives a sense of ownership to the developers of the infrastructure for the services, via the definitions in the Docker Compose file that defines the application stack resources and services along with resources access and permissions.

CHAPTER

FORTYSEVEN

WHY DID I CREATE ECS COMPOSE-X?

Many companies I have worked with struggle with providing a true cloudy experience to their developers and enable them to deploy AWS resources in a controlled fashion. And when they do give poweruser/administrator level of permissions to developers, they usually have not been trained appropriately to understand fundamentals, such as least privileges and you end up with services which all use the same AWS Access and Secret keys (yes, I witnessed it recently) and these keys stay around for eternity (seen 1000+ days).

As an AWS Cloud Engineer, this scares the hell out of me and I feel like this is the first thing I need to fix. As an automation engineer, I wanted a tool that allows developers to keep using Docker compose, as they very often do, so they can't run their workload on their laptops for quick testing and application testing.

But, "It works on my laptop" is something that in 2020 is simply unacceptable to companies deploying microservices.

Therefore, combining my love for least privileges and therefore IAM instance capability to implement it, and the need for a tool going these extra miles, I decided to simply go for it.

A lot of you probably would prefer to use some other tools, such as Terraform. But I all heartily believe that cloud engineers should use the IaC provided by the Cloud provider.

Third party integrations are coming, including for example the excellent AWS CFN registries where we already see partners like DataDog provide the ability to create non AWS resources as part of the CFN stack and remove the need for custom made code.

CHAPTER

FORTYEIGHT

WHY AM I NOT USING AWS CDK?

ECS Compose-X was started before AWS CDK came out with any python support, and python was the language of choice for this project.

Therefore, Troposphere was the obvious choice as the python library to use to build all the CFN templates. The way Troposphere has been built is simple and clear, the name of the properties are the same as they are in AWS CloudFormation, which gives a sense of standard to the user, allowing an experience as close to copy-paste as possible.

Troposphere has a very strong community and has wide set of AWS services support. The community is active and other AWS Projects members are directly involved in the day-to-day life of the project.

In CDK, all the properties you have to set for a CFN resource have been renamed, Troposphere kept the same name definition for the resources properties. To me, this is a very valuable thing, not to have to map CFN properties to a language specific one.

CHAPTER

FORTYNINE

IMPLEMENTING LEAST PRIVILEGES AT THE HEART OF ECS COMPOSE-X

One of the most important value add for a team of Cloud/DevOps engineers who have to look after an environment to use ECS Compose-X is the persistent implementation of best practices:

- All microservices are using different sets of credentials
- All microservices are isolated by default and allowed traffic only when explicitly permitted
- All microservices must be defined as the consumer of a resource (DB, Queue, Table) to be granted access to it.

There have been to many instances of breaches on AWS due to a lack of strict IAM definitions and permissions. Automation can solve that problem and with ECS Compose-X the effort is to constantly abide by the least privileges access principle.

CHAPTER

FIFTY

CONTRIBUTORS

- **John Preston**
    - Github
    - Keybase

CHAPTER

FIFTYONE

CREDITS

This package would not have been possible without the amazing job done by the AWS CloudFormation team! Thank you to all people working on their awesome libaries, to name a few:

- Troposphere
- placebo
- behave

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

FIFTYTWO

INDICES AND TABLES

- genindex
- modindex
- search