Handel Documentation

Release 0.23.2

David Woodruff

Dec 20, 2018

Getting Started

Handel is a library that orchestrates your AWS deployments so you don't have to.

Handel is built on top of CloudFormation with an aim towards easier AWS provisioning and deployments. You give Handel a configuration file (the *Handel file*) telling it what services you want in your application, and it wires them together for you.

Here's an example Handel file defining a Beanstalk application to be deployed with an SQS queue and S3 bucket:

```
version: 1

name: my-first-handel-app

environments:
  dev:
    webapp:
      type: beanstalk
      path_to_code: .
      solution_stack: 64bit Amazon Linux 2017.09 v4.4.5 running Node.js
      dependencies:
      - bucket
      - queue
    bucket:
      type: s3
    queue:
      type: sqs
```

From this Handel file, Handel creates the appropriate CloudFormation templates for you, including taking care of all the tricky security bits to make the services be able to talk to each other.

CHAPTER 1

Introduction

Handel is a CLI tool that will help you more easily deploy your application to AWS. You specify a declarative file in your application called *handel.yml*, and Handel will deploy your application to AWS for you.

Handel runs on top of CloudFormation. It automatically creates CloudFormation templates from your Handel file, and deploys your applications in a secure fashion, providing a vastly easier experience than using vanilla CloudFormation.

1.1 Why does Handel exist?

Handel runs on top of CloudFormation, so why not use CloudFormation directly?

The main answer is that using CloudFormation comes with a very steep learning curve. The main difficulty comes not in learning the configuration language itself, but much more in the interactions required between resources with IAM roles and EC2 security groups.

By running on top of CloudFormation, Handel provides the following benefits:

- Automatic security wiring, freeing you from having to worry about EC2 security groups and IAM roles.
- Much simpler interface to configuring an application. A 400-line CloudFormation template can be configured in more like 30-40 lines. See *Handel vs. CloudFormation* for an example of this.

By using Handel, you get to retain the benefits of CloudFormation with less work!

1.2 What AWS services are supported?

See the *Supported Services* section for information on which AWS services you can currently use with Handel.

1.3 How can I deploy an application with Handel?

First, see the *Installation* section to install Handel.

After you've installed Handel, see the *Creating Your First Handel App* page for a tutorial on creating a simple app and deploying it with Handel.

CHAPTER 2

Handel vs. CloudFormation

CloudFormation is one of the most commonly used methods for automatically deploying applications to AWS. In fact, Handel uses CloudFormation under the hood to do your deployments. This page compares using vanilla CloudFormation and the Handel library.

2.1 CloudFormation

CloudFormation is one of the most popular ways to deploy applications to AWS. It is an extremely flexible tool that allows you great control over how you wire up applications. That flexibility comes at the cost of complexity. You need to learn quite a bit before you can ever deploy your first production-quality application.

Here is an example CloudFormation template that creates a Beanstalk server and wires it up with an S3 bucket, a DynamoDB table, and an SQS queue:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Beanstalk application with SQS queue, S3 bucket, and DynamoDB table

Resources:
  Queue:
    Type: AWS::SQS::Queue
    Properties:
      DelaySeconds: 0
      MaximumMessageSize: 262144
      MessageRetentionPeriod: 345600
      QueueName: dsw88-testapp-dev-queue-sqs
      ReceiveMessageWaitTimeSeconds: 0
      VisibilityTimeout: 30

  Table:
    Type: "AWS::DynamoDB::Table"
    Properties:
      AttributeDefinitions:
      - AttributeName: MyPartitionKey
```

(continues on next page)

(continued from previous page)

```
      AttributeType: S
    KeySchema:
    - AttributeName: MyPartitionKey
      KeyType: HASH
    ProvisionedThroughput:
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
    TableName: dsw88-testapp-dev-table-dynamodb

Bucket:
  Type: "AWS::S3::Bucket"
  Properties:
    BucketName: dsw88-testapp-dev-bucket-s3
    VersioningConfiguration:
      Status: Enabled

BeanstalkRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
      - Sid: ''
        Effect: Allow
        Principal:
          Service: ec2.amazonaws.com
        Action: sts:AssumeRole
    Path: /services/
    RoleName: dsw88-testapp-dev-webapp-beanstalk

BeanstalkPolicy:
  Type: AWS::IAM::Policy
  Properties:
    PolicyDocument:
      Version: '2012-10-17'
      Statement:
      - Effect: Allow
        Action:
        - s3:ListBucket
        Resource:
        - arn:aws:s3:::dsw88-testapp-dev-bucket-s3
      - Effect: Allow
        Action:
        - s3:PutObject
        - s3:GetObject
        - s3:DeleteObject
        Resource:
        - arn:aws:s3:::dsw88-testapp-dev-bucket-s3/*
      - Effect: Allow
        Action:
        - sqs:ChangeMessageVisibility
        - sqs:ChangeMessageVisibilityBatch
        - sqs:DeleteMessage
        - sqs:DeleteMessageBatch
        - sqs:GetQueueAttributes
        - sqs:GetQueueUrl
        - sqs:ListDeadLetterSourceQueues
```

(continues on next page)

(continued from previous page)

```
          - sqs:ListQueues
          - sqs:PurgeQueue
          - sqs:ReceiveMessage
          - sqs:SendMessage
          - sqs:SendMessageBatch
          Resource:
          - arn:aws:sqs:us-west-2:111111111111:dsw88-testapp-dev-queue-sqs
        - Sid: DyanmoDBAccessT7eFcR52BF7VnlQF
          Effect: Allow
          Action:
          - dynamodb:BatchGetItem
          - dynamodb:BatchWriteItem
          - dynamodb:DeleteItem
          - dynamodb:DescribeLimits
          - dynamodb:DescribeReservedCapacity
          - dynamodb:DescribeReservedCapacityOfferings
          - dynamodb:DescribeStream
          - dynamodb:DescribeTable
          - dynamodb:GetItem
          - dynamodb:GetRecords
          - dynamodb:GetShardIterator
          - dynamodb:ListStreams
          - dynamodb:PutItem
          - dynamodb:Query
          - dynamodb:Scan
          - dynamodb:UpdateItem
          Resource:
          - arn:aws:dynamodb:us-west-2:111111111111:table/dsw88-testapp-dev-table-
↪dynamodb
        - Sid: BucketAccess
          Action:
          - s3:Get*
          - s3:List*
          - s3:PutObject
          Effect: Allow
          Resource:
          - arn:aws:s3:::elasticbeanstalk-*
          - arn:aws:s3:::elasticbeanstalk-*/*
        - Sid: XRayAccess
          Action:
          - xray:PutTraceSegments
          - xray:PutTelemetryRecords
          Effect: Allow
          Resource: "*"
        - Sid: CloudWatchLogsAccess
          Action:
          - logs:PutLogEvents
          - logs:CreateLogStream
          Effect: Allow
          Resource:
          - arn:aws:logs:*:*:log-group:/aws/elasticbeanstalk*
        - Sid: ECSAccess
          Effect: Allow
          Action:
          - ecs:Poll
          - ecs:StartTask
          - ecs:StopTask
```

(continues on next page)

(continued from previous page)

```
          - ecs:DiscoverPollEndpoint
          - ecs:StartTelemetrySession
          - ecs:RegisterContainerInstance
          - ecs:DeregisterContainerInstance
          - ecs:DescribeContainerInstances
          - ecs:Submit*
          - ecs:DescribeTasks
          Resource: "*"
      PolicyName: dsw88-testapp-dev-webapp-beanstalk
      Roles:
      - !Ref BeanstalkRole

  InstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Path: "/services/"
      Roles:
      - !Ref BeanstalkRole

  BeanstalkSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: dsw88-testapp-dev-webapp-beanstalk
      VpcId: vpc-aaaaaaaa
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '22'
        ToPort: '22'
        SourceSecurityGroupId: sg-44444444
      SecurityGroupEgress:
      - IpProtocol: tcp
        FromPort: '0'
        ToPort: '65335'
        CidrIp: 0.0.0.0/0
      Tags:
      - Key: Name
        Value: dsw88-testapp-dev-webapp-beanstalk

  BeanstalkIngressToSelf:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId:
        Ref: BeanstalkSecurityGroup
      IpProtocol: tcp
      FromPort: '0'
      ToPort: '65335'
      SourceSecurityGroupId:
        Ref: BeanstalkSecurityGroup

  Application:
    Type: AWS::ElasticBeanstalk::Application
    Properties:
      ApplicationName: dsw88-testapp-dev-webapp-beanstalk
      Description: Application for dsw88-testapp-dev-webapp-beanstalk

  ApplicationVersion:
    Type: AWS::ElasticBeanstalk::ApplicationVersion
```

(continues on next page)

(continued from previous page)

```
    Properties:
      ApplicationName: !Ref Application
      Description: Application version for dsw88-testapp-dev-webapp-beanstalk
      SourceBundle:
        S3Bucket: beanstalk-us-west-2-111111111111
        S3Key: dsw88-testapp/dev/webapp/beanstalk-deployable-SOME_GUID.zip

  ConfigurationTemplate:
    DependsOn:
    - Queue
    - Table
    - Bucket
    - BeanstalkSecurityGroup
    - InstanceProfile
    Type: AWS::ElasticBeanstalk::ConfigurationTemplate
    Properties:
      ApplicationName: !Ref Application
      Description: Configuration template for dsw88-testapp-dev-webapp-beanstalk
      OptionSettings:
      - Namespace: aws:autoscaling:launchconfiguration
        OptionName: IamInstanceProfile
        Value: !Ref InstanceProfile
      - Namespace: aws:autoscaling:asg
        OptionName: MinSize
        Value: 1
      - Namespace: aws:autoscaling:asg
        OptionName: MaxSize
        Value: 1
      - Namespace: aws:autoscaling:launchconfiguration
        OptionName: InstanceType
        Value: t2.micro
      - Namespace: aws:autoscaling:launchconfiguration
        OptionName: SecurityGroups
        Value: !Ref BeanstalkSecurityGroup
      - Namespace: aws:autoscaling:updatepolicy:rollingupdate
        OptionName: RollingUpdateEnabled
        Value: true
      - Namespace: aws:ec2:vpc
        OptionName: VPCId
        Value: vpc-aaaaaaaa
      - Namespace: aws:ec2:vpc
        OptionName: Subnets
        Value: subnet-ffffffff,subnet-77777777
      - Namespace: aws:ec2:vpc
        OptionName: ELBSubnets
        Value: subnet-22222222,subnet-66666666
      - Namespace: aws:ec2:vpc
        OptionName: DBSubnets
        Value: subnet-eeeeeeee,subnet-cccccccc
      - Namespace: aws:ec2:vpc
        OptionName: AssociatePublicIpAddress
        Value: false
      - Namespace: aws:elasticbeanstalk:application:environment
        OptionName: MY_INJECTED_VAR
        Value: myValue
      SolutionStackName: 64bit Amazon Linux 2016.09 v4.0.1 running Node.js
```

(continues on next page)

(continued from previous page)

```
  Environment:
    Type: "AWS::ElasticBeanstalk::Environment"
    Properties:
      ApplicationName: !Ref Application
      Description: environment for dsw88-testapp-dev-webapp-beanstalk
      TemplateName: !Ref ConfigurationTemplate
      VersionLabel: !Ref ApplicationVersion
      Tags:
      - Key: Name
        Value: dsw88-testapp-dev-webapp-beanstalk

Outputs:
  BucketName:
    Description: The endpoint URL of the beanstalk environment
    Value:
      Fn::GetAtt:
        - Environment
        - EndpointURL
```

2.2 Handel

Handel is a deployment library that runs on top of CloudFormation. The services you specify in Handel are turned into CloudFormation templates that are created on your behalf.

Because of this approach, Handel frees you from having to worry about the detail of CloudFormation, as well as security services such as IAM and VPC. This simplicity comes at the cost of lack of flexibility in some cases. For example, when wiring up permissions between a Beanstalk app and an S3 bucket, you don't get to choose what permissions exactly will be applied. Handel will apply what it considers to be reasonable and secure permissions.

Here is an example Handel file that creates the same set of resources (Beanstalk, S3, DynamoDB, and SQS) as the CloudFormation template above:

```
version: 1

name: dsw88-testapp

environments:
  dev:
    webapp:
      type: beanstalk
      path_to_code: .
      solution_stack: 64bit Amazon Linux 2016.09 v4.0.1 running Node.js
      instance_type: t2.micro
      health_check_url: /
      min_instances: 1
      max_instances: 1
      environment_variables:
        MY_INJECTED_VAR: myValue
      dependencies:
      - bucket
      - queue
      - table
    bucket:
      type: s3
```

(continues on next page)

(continued from previous page)

```
  queue:
    type: sqs
  table:
    type: dynamodb
    partition_key:
      name: MyPartionKey
      type: String
    provisioned_throughput:
      read_capcity_units: 1
      write_capacity_units: 1
```

Note the greatly reduced file size, as well as the lack of any IAM or VPC configuration details.

CHAPTER 3

Installation

Handel is a CLI tool written in Node.js. In order to install it, you will first need Node.js installed on your machine.

3.1 Installing Node.js

The easiest way to install Node.js is to download the compiled binaries from the Node.js website. Handel requires Node.js *version 6.x or greater* in order to run.

Once you have completed the installation on your machine, you can verify it by running these commands:

```
node --version
npm --version
```

The above commands should show you the versions of Node and NPM, respectively.

3.2 Installing Handel

Once you have Node.js installed, you can use the NPM package manager that is bundled with Node.js to install Handel:

```
npm install -g handel
```

When the above commands complete successfully, you should be able to run the Handel CLI to deploy your application.

3.3 Next Steps

See the *Creating Your First Handel App* section for a tutorial on deploying a simple Node.js application to AWS using Handel.

CHAPTER 4

Creating Your First Handel App

This page contains a tutorial for writing a simple Node.js "Hello World!" app and deploying it to AWS with the Handel tool.

Important: Before going through this tutorial, make sure you have installed Handel on your machine as shown in the *Installation* section.

4.1 Tutorial

This tutorial contains the following steps:

1. *Write the app*
2. *Create your Handel file*
3. *Deploy using Handel*
4. *Delete the created app*

Follow along with each of these steps in the sections below in order to complete the tutorial.

4.1.1 Write the app

We first need to create an app that you can run. We're going to use Node.js to create an Express web service that will run in ElasticBeanstalk.

First create a directory for your application code:

```
mkdir my-first-handel-app
cd my-first-handel-app
```

Since it's a Node.js application, the first thing you'll need is a package.json file that specifies information about your app, including its dependncies. Create a file named *package.json* with the following contents:

```
{
    "name": "my-first-handel-app",
    "version": "0.0.1",
    "author": "David Woodruff",
    "dependencies": {
        "express": "^4.15.2"
    }
}
```

Now that you've got your package.json, install your dependencies from NPM:

```
npm install
```

Next, create a file called *app.js* with the following contents:

```
var app = require('express')();

app.get('/', function(req, res) {
    res.send("Hello World!");
});

var port = process.env.PORT || 3000;
app.listen(port, function () {
    console.log('Server running at http://127.0.0.1:' + port + '/');
});
```

Note: The above app code uses Express to set up a web server that has a single route "/". That route just responds with the string "Hello World!".

Test your app by starting it up:

```
node app.js
```

Once it's started up, you should be able to go to http://localhost:3000/ to see it working. You should see a page that says "Hello World!" on it.

4.1.2 Create your Handel file

Now that you've got a working app, you need to create a Handel file specifying how you want your app deployed. Create a file called *handel.yml* with the following contents:

```
version: 1

name: my-first-handel-app # This is a string you choose for the name of your app.

environments:
  dev: # This is the name of your single environment you specify.
    webapp: # This is the name of your single service inside your 'dev' environment.
      type: beanstalk # Every Handel service requires a 'type' parameter
      path_to_code: . # This contains the path to the directory where your code lives␣
↪that should be sent to Beanstalk
      solution_stack: 64bit Amazon Linux 2018.03 v4.5.0 running Node.js # This␣
↪specifies which Beanstalk 'solution stack' should be used for the app.
```

Note: See the *Handel File* section for full details on how the Handel file is structured.

Note: We only specified the required parameters for Beanstalk. There are others that have defaults if you don't specify them. See the *Beanstalk* service documentation for full information on all the different parameters for the service.

4.1.3 Deploy using Handel

Important: In order to run Handel to deploy your app, you must be logged into your AWS account on the command line. You can do this by setting your AWS access keys using the AWS CLI.

See Configuring the AWS CLI for help on doing this once you've installed the AWS CLI.

If you work for an organization that uses federated logins through something like ADFS, then you'll have a different process for logging in on the command-line. In this case, ask your organization how they login to AWS on the command-line.

Now that you've written your app, created your Handel file, and obtained your account config file, you can run Handel to deploy:

```
handel deploy -c default-us-east-1 -e dev
```

Note: In the above command, the following arguments are provided:

- The *-c* parameter specifies which *Account Config File* to use. Specifying *default-us-east-1* here tells Handel you don't have one and just want to use the default VPC AWS provides in the us-east-1 region.
- The *-e* parameter is a comma-separated string list that specifies which environments from your Handel file you want to deploy

Once you've executed that command, Handel should start up and deploy your application. You can sign into the AWS Console and go to the "ElasticBeanstalk" service to see your deployed application.

4.1.4 Delete the created app

Since this was a tutorial using a *Hello World* app, we want to delete it now that we're done with it. To delete your app, run the following command:

```
handel delete -c default-us-east-1 -e dev
```

When you execute the above command, it will show you something like this confirmation prompt:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WARNING: YOU ARE ABOUT TO DELETE YOUR HANDEL ENVIRONMENT 'dev'!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

If you choose to delete this environment, you will lose all data stored in the␣
↪environment!
```

(continues on next page)

(continued from previous page)

```
In particular, you will lose all data in the following:

* Databases
* Caches
* S3 Buckets
* EFS Mounts

PLEASE REVIEW this environment thoroughly, as you are responsible for all data loss␣
↪associated with an accidental deletion.
PLEASE BACKUP your data sources before deleting this environment just to be safe.

? Enter 'yes' to delete your environment. Handel will refuse to delete the␣
↪environment with any other answer:
```

Type *yes* and hit *Enter*, and Handel will proceed to delete the environment.

Congratulations, you've finished the tutorial!

4.2 Next Steps

Now that you've deployed a simple app using Handel, where do you go next?

4.2.1 Learn more about Handel

Read through the following documents in the *Handel Basics* section:

- *Handel File*
- *Service Dependencies*
- *Consuming Service Dependencies*
- *Service Events*

Those documents will give you more information on the design and usage of Handel, particularly how you can use Handel's dependencies to wire services together.

4.2.2 Learn how to configure the different service types

Once you understand Handel's basic configuration, see the *Supported Services* section, which contains information about the different services you can deploy using Handel. Each service page in that section will give the following information:

- How to configure the service in your Handel file.
- How to consume the service in other services (if applicable).
- How to produce events to other services (if applicable).

4.2.3 Set up a continuous delivery pipeline

Handel can run anywhere, from your laptop to a build server. The recommended way to run Handel is inside a Continuous Delivery pipeline. There are many options available on the market, and AWS provides the CodePipeline service for creating these pipelines.

Handel provides a companion tool, called Handel-CodePipeline, that helps you easily create these CodePipelines running Handel for your deploy.

CHAPTER 5

CLI Reference

The Handel command-line interface should be run in a directory with a *handel.yml* file.

It defines three commands: *check*, *deploy*, and *delete*

5.1 *handel check*

Validates that a given Handel configuration is valid.

Note that this does not validate against account-level settings, such as *Requiring Tags*.

5.1.1 Parameters

handel check does not accept parameters.

5.2 *handel deploy*

Validates and deploys the resources in a given environment.

5.2.1 Parameters

Parameter	Type	Required	Default	Description
-c <value>	string	Yes		Path to account config or base64 encoded JSON string of config
-e <env>[,<env>]	comma-separated list	Yes		List of environments from the handelfile to deploy.
-d	boolean (present or not present)	No	false	If set, turns on debug-level logging.
-t <key>=<value>[,<key>=<value>]	comma-separated list of key-value pairs	No		List of tags to apply to all resources in the handelfile. These override any static tags set in the handelfile.

5.3 *handel delete*

Deletes all resources in a given environment.

5.3.1 Parameters

Parameter	Type	Required	Default	Description
-c <value>	string	Yes		Path to account config or base64 encoded JSON string of config
-e <env>[,<env>]	comma-separated list	Yes		List of environments from the handelfile to delete.
-d	boolean (present or not present)	No	false	If set, turns on debug-level logging.
-y	boolean (present or not present)	No	false	If set, Handel will *not* prompt for confirmation of the delete action.

CHAPTER 6

Handel File

In order to provide Handel with the information it needs to deploy your services, you must create a YAML configuration file for your application. This file must be named *handel.yml*. This page contains information on the structure of that file.

6.1 Terminology

Handel uses the following terminology in the context of the Handel file:

Application In Handel, an *application* is a logical container for of all the resources specified in your Handel file. This application is composed of one or more *environments*.

Environment An *environment* is a collection of one or more AWS services that form a single unit intended for use together. This construct allows you to have multiple instances of your application running in different configurations.

Many applications, for example, have a 'dev' environment for testing new changes, and a 'prod' environment for the actual production application that end-users hit. There are many other possible environments that an application may define.

Each environment you specify constitutes a single instance of your application configured in a certain way.

Service In an environment, a *service* is a single Handel service that is deployed via a CloudFormation stack. This service takes configuration parameters to determine how to deploy it. It can also reference other services in your environment that it depends on at runtime. Handel will auto-wire these services together for you and inject their information into your application.

6.2 Handel File Specification

The Handel file is a YAML file that must conform to the following specification:

```
version: 1

name: <name of the app being deployed>

tags:
  tag-name: value

environments:
  <environment_name>:
    <service_name>:
      type: <service_type>
      <service_param>: <param_value>
      dependencies:
      - <service name>
```

6.2.1 Handel File Explanation

name The name field is the top-level namespace for your application. This field is used in the naming of virtually all your AWS resources that Handel creates.

<environment_name> The <environment_name> key is a string you provide to specify the name of an environment. You can have multiple environments in your Handel application. This environment field is used in the naming of virtually all your AWS resources that Handel creates.

<service_name> The <service_name> key is a string you provide to specify the name of a Handel service inside an environment. You can have multiple services in an environment. This service field is used in the naming of virtually all your AWS resources that Handel creates.

dependencies In a given Handel service, you can use the 'dependencies' field to specify other services in your environment with which your service needs to communicate.

Note: Not all AWS services can depend on all other AWS services. You will get an error if you try to depend on a service that is not consumable by your service.*

6.2.2 Limits

The following limits exist on names in the Handel file:

Element	Length Limit	Allowed Characters
name	30 characters	Alphanumeric (a-Z, 0-9) and dashes (-)
<environment_name>	10 characters	Alphanumeric (a-Z, 0-9) and dashes (-)
<service_name>	20 characters	Alphanumeric (a-Z, 0-9) and dashes (-)

There may be other service-specific limits. See *Supported Services* for information on service-specific limits.

CHAPTER 7

Account Config File

Handel requires two pieces of information in order to deploy your application:

- Your handel.yml file that contains your service specification
- Account configuration information that contains items like VPCs and subnets to use when deploying applications.

You can either choose to let Handel just use the AWS default VPC, or you can provide it with an Account Config File that contains the information about your own custom VPC to use.

Important: If you're running Handel inside a company or organization AWS account, it is likely your company has already set up VPCs how they want them. In this case, get your platform/network group to help you configure this account config file for your VPC.

7.1 Using the AWS default VPC

If you're using Handel in a personal AWS account, it's likely that you don't want to have to set up a VPC and create your own account config file. In this case, Handel can just use the default VPC that AWS provides. You tell Handel to use these defaults in this way:

```
handel deploy -c default-us-east-1 -e dev
```

Notice that in the *-c* parameter, we are passing the string *default-us-east-1*, which tells Handel to use the default VPC in the us-east-1 region.

Note: To use a default VPC, specify it with the following pattern:

```
default-<region>
```

The <region> parameter is the name of the AWS region, such as *us-east-1* or *us-west-2*, where you want to run your app.

7.2 Using Handel at a company or organization

It is best if someone with a knowledge of the account-level network configuration creates this account configuration file. This file can then be shared by all services that deploy in that account.

If you're using Handel in a company or organization account, talk to your platform/network group that administers the VPCs in your account. They can help you know what values to put in your account config file.

7.3 Account Config File Specification

The account config file is a YAML file that must contain the following information:

```
account_id: <string> # Required. The numeric ID of your AWS account.
region: <string> # Required. The region, such as 'us-west-2' that your VPC resides in.
vpc: <string> # Required. The ID of your VPC in which to deploy your applications.
public_subnets: # Required. A list of one or more subnet IDs from your VPC where you␣
↪want to deploy publicly available resources.
- <string>
private_subnets: # Required. A list of one or more subnet IDs from your VPC where you␣
↪want to deploy private resources.
- <string>
data_subnets: # Required. A list of one or more subnet IDs from your VPC where you␣
↪want to deploy databases (such as RDS and ElastiCache)
- <string>
ssh_bastion_sg: <string> # The ID of the security group you
elasticache_subnet_group: <string> # The name of the ElastiCache subnet group to use␣
↪when deploying ElastiCache clusters.
rds_subnet_group: <string> # The name of the RDS subnet group to use when deploying␣
↪RDS clusters.
required_tags: # Optional. Allows an organization to enforce rules about tagging␣
↪resources. This is a list of tag names that must be set on each Handel application␣
↪or resource.
- <string>
handel_resource_tags: # Optional. Sets tags to be applied to any generic resources,␣
↪such as lambda functions, that Handel uses internally.
  <key>: <value>
  <key>: <value>
```

Important: Be sure to put quotes around the *account_id* field in your account config file!

If you dont, YAML will treat it as a number. This can cause problems if your account ID starts with a *0*, because the JavaScript YAML parser that Handel uses will parse it as an octal number, resulting in a totally different account ID.

CHAPTER 8

Service Dependencies

One of the key features of Handel is being able to configure an AWS service such as Beanstalk to depend on another AWS service such as DynamoDB. Rather than having to figure out the security interactions between the two, Handel will auto-wire the services together for you.

8.1 Specifying Dependencies

To specify a dependency on a service, add a 'dependencies' list in your service definition with the list values being the service names of the services you wish to consume. The following example shows a Beanstalk service specifying a dependency on an SQS queue:

```
version: 1

name: beanstalk-example

environments:
  dev:
    webapp:
      type: beanstalk
      path_to_code: .
      solution_stack: 64bit Amazon Linux 2016.09 v4.0.1 running Node.js
      instance_type: t2.micro
      health_check_url: /
      min_instances: 1
      max_instances: 1
      dependencies:
      - queue
    queue:
      type: sqs
```

Important: Notice that the item in the dependencies list called *queue* is referring to the service name specified for

the SQS queue.

See *Consuming Service Dependencies* for information about how your consuming app (such as Beanstalk) can get the information it needs to talk to your service dependency (such as SQS).

CHAPTER 9

Consuming Service Dependencies

When you specify a dependency on a service using *Service Dependencies*, that service is auto-wired to your application. This page contains information about how you can consume those injected dependencies in your application code to actually communicate with these services.

When Handel wires services together securely, it will inject environment variables into the consuming service for each service that it depends on. These environment variables provide information about the created service that tell you information such as where to find the service and how to communicate with it.

The following Handel file defines a Beanstalk service that depends on an SQS queue:

```
version: 1

name: beanstalk-example

environments:
  dev:
    webapp:
      type: beanstalk
      path_to_code: .
      solution_stack: 64bit Amazon Linux 2016.09 v4.0.1 running Node.js
      instance_type: t2.micro
      health_check_url: /
      min_instances: 1
      max_instances: 1
      dependencies:
      - my-queue
    my-queue:
      type: sqs
```

Handel will inject environment variables in the Beanstalk application for the SQS queue, such as the queue's ARN, name, and URL. You can read these environment variables when you are writing code to communicate with the queue.

9.1 Environment Variable Names

Every environment variable injected by Handel for service dependencies has a common structure.

This environment variable name consists of the dependency's name (as defined in the Handel file), followed by the name of the value being injected.

```
<SERVICE_NAME>_<VALUE_NAME>
```

In the above example, the referencing Beanstalk application would need to use the following name to get the URL of the SQS Queue:

```
MY_QUEUE_QUEUE_URL
```

Note: All Handel injected environment variables will be all upper-cased, with dashes converted to underscores.

9.2 Parameter Store Prefix

Handel puts auto-generated credentials and other secrets in the EC2 Parameter Store, and it wires up your applications to allow you to access these secrets.

Each parameter Handel puts in the parameter store has a common prefix, which is defined by the following structure:

```
<app_name>.<environment_name>
```

You can use the *Common Injected Environment Variables* to obtain the value of this prefix.

9.3 Common Injected Environment Variables

In addition to environment variables injected by services your applications consume, Handel will inject a common set of environment variables to all applications:

Environment Variable	Description
HANDEL_APP_NAME	This is the value of the *name* field from your Handel file. It is the name of your application.
HANDEL_ENVIRONMENT_NAME	This is the value of the <*environment*> field from your Handel file. It is the name of the environment the current service is a part of.
HANDEL_SERVICE_NAME	This is the value of the <*service_name*> field from your Handel file. It is the name of the currently deployed service.
HANDEL_PARAMETER_STORE_PREFIX	This is the *prefix* used for secrets stored in Parameter Store.
HANDEL_REGION_NAME	This is the value of the <*region_name*> field from your Handel file, or the current region if the region id not specified.

CHAPTER 10

Service Events

Many AWS services are able to send *events* to other AWS services. For example, the S3 service can send events about file changes in a bucket to another service such as Lambda.

Handel allows you to specify event consumers for a particular service in your Handel file. Handel will then perform the appropriate wiring on both services to configure the producer service to send events to the consumer service.

10.1 Specifying Service Events

To configure service events on a particular Handel service, add an *event_consumers* list in your producer service definition. This list contains information about the services that will be consuming events from that producer service.

The following example shows an SNS topic specifying producing events to an SQS queue:

```
version: 1

name: sns-events-example

environments:
  dev:
    topic:
      type: sns
      event_consumers:
      - service_name: queue
    queue:
      type: sqs
```

When you specify event consumers in your producer service, you don't need to specify anything on the consumer services. They will be automatically wired appropriately to the producer service in which you specified them as consumers.

Note: Not all services may produce events, and not all services may consume events. You will get an error if you try

to specify a producer or consumer service that don't support events.

CHAPTER 11

Accessing Application Secrets

Many applications have a need to securely store and access *secrets*. These secrets include things like database passwords, encryption keys, etc. This page contains information about how you can store and access these secrets in your application when using Handel.

> **Warning:** **Do not** pass these secrets into your application as environment variables in your Handel file. Since you commit your Handel file to source control, any credentials you put in there would be compromised to anyone who can see your source code.
>
> Handel provides a different mechanism for passing secrets to your application, as explained in this document.

11.1 Application Secrets in Handel

Handel uses the EC2 Systems Manager Parameter Store for secrets storage. This service provides a key/value store where you can securely store secrets in a named parameter. You can then call the AWS API from your application to obtain these secrets.

Handel automatically wires up access to the Parameter Store in your applications, granting you access to get parameters whose names start with a particular path. Handel wires up permissions for parameters with the following path:

```
/<appName>/<environmentName>/
```

To see a concrete illustration of this, consider the following example Handel file, which defines a single Lambda:

```
version: 1

name: my-lambda-app

environments:
  dev:
    function:
```

(continues on next page)

(continued from previous page)

```
    type: lambda
    path_to_code: .
    handler: app.handler
    runtime: nodejs6.10
```

This Lambda, when deployed, will be able to access any EC2 Parameter Store parameters under the path "/my-lambda-app/dev/". Thus, the parameter `/my-lambda-app/dev/somesecret` would be available to this application, but the `/some-other-app/dev/somesecret` parameter would not, because it is not included in the same path.

Note: As a convenience, Handel injects an environment variable called `HANDEL_PARAMETER_STORE_PATH` into your application. This variable contains the pre-built `/<appName>/<environmentName>/` path so that you don't have to build it yourself.

Warning: Previously Handel wired permmissions based on a prefix like: `<appName>.<environmentName>` This functionality is being deprecated in favor of paths. As a convenience, Handel still wires the permissions and injects an environment variable called `HANDEL_PARAMETER_STORE_PREFIX` into your application. This variable contains the pre-built `<appName>.<environmentName>` prefix so that you don't have to build it yourself. Please only use prefix if required. Otherwise Path is preferred. More info can be found Here

11.1.1 Global Parameters

It is a common desire to share some parameters globally with all apps living in an account. To support this, Handel also grants your application permission to access a special global namespace of parameters that start with the following prefix:

```
handel.global
```

Parameters that start with this prefix are available to any app deployed using Handel in the account and region that you're running in.

Warning: Any parameter you put here WILL be available to any other user of Handel in the account. Don't put secrets in this namespace that belong to just your app!

11.2 Adding a Parameter to the Parameter Store

See the Walkthrough in the AWS documentation for an example of how to add your parameters.

Important: When you add your parameter, remember to start the name of the parameter with your application name from your Handel file.

11.3 Getting Parameters from the Parameter Store

Once you've added a parameter to the Parameter Store with the proper prefix, your deployed application should be able to access it. See the example of CLI access for the get-parameters call in the Walkthrough for information on how to do this.

The example in the walkthrough shows an example using the CLI, but you can use the AWS language SDKs with the getParameters call in a similar manner. See the documentation of the SDK you are using for examples.

CHAPTER 12

Tagging

Most AWS services support the tagging of resources. You can use tags to apply arbitrary metadata to AWS resources. This metadata is available with the resources, and can be used for a variety of purposes. Here are some examples of what you can use tags for:

- Generating cost-utilization reports.
- Providing information about teams developing the product such as contact information.
- Specifying which resources may be automatically shut down or terminated by an external script.

AWS services have limits on the total number of tags that may be applied to each service. As of January 2018, most services have a limit of 50 tags.

12.1 Application Tags

In your handel.yml file, you can specify tags that apply to all supported resources in the stack, as well as the underlying Cloudformation stacks. You can specify these tags using a top-level 'tags' object:

```
version: 1

name: <name of the app being deployed>

tags:
  your-tag: value
  another-tag: another value
  technical-owner: Joe Developer <joe_developer@example.com>
  business-owner: Jill Manager <jill_manager@example.com>

environments:
  ...
```

12.2 Resource Tags

On resources that support it, Handel allows you to specify tags for that resource. It will make the appropriate calls on your behalf to tag the resources it creates with whatever tags you choose to apply.

Resource-level tags will override any application-level tags with the same name.

Resource-level tags are defined by the following schema:

```
environments:
  my-service:
    type: foo-service
    tags:
      yourtag: value
      another-tag: another value
```

12.2.1 Tagging Unsupported Resources

Attention: Some AWS resource types do not support tagging. In these cases, any related, taggable resources will be tagged, as will the Cloudformation stack that Handel uses to provision the resources.

If AWS adds tagging support to any of these services, the next Handel deploy should result in tags automatically being applied to the resources by Cloudformation.

Example: ECS does not currently support tagging Task Definitions. Handel will, however, tag any Application Load Balancers that are provisioned to service that ECS configuration, as well as the Cloudformation stack that provisioned them.

12.3 Default Tags

In addition to the ones you specify yourself, Handel will automatically apply the following tags to your AWS resources:

- *app* - This will contain the value from the *name* field in your Handel file, which is the name of your overall application.
- *env* - This will contain the value of the <*environment_name*> that your service is a part of.

See *Handel File Explanation* for a refresher on where these automatically applied values fit in your Handel file.

12.4 Requiring Tags

Some organizations may wish to enforce a specific resource tagging scheme. For example, in addition to Handel's *app* and *env* tags, they may wish to require that all resource have a *technical-owner* and *business-owner* tag.

Tag requirements can be configured in the *Account Config File*. If a user attempts to deploy an application that does not define the required tags either at the application level or the resource level, the deployment will fail.

CHAPTER 13

Deleting an Environment

Once you've created an application using Handel, you may decide to delete one or more of your environments. This document tells how to delete your environments.

> **Danger:** If you delete an environment, it will delete all data in your environment!
>
> Please review the data in an environment carefully before deleting it! Handel just helps you create and delete your resources, you are responsible for making sure you don't delete resources you care about.

Execute Handel's delete lifecycle at the command line. Here is an example of deleting an environment:

```
# Make sure to replace the *-c* and *-e* flags in the below command with the correct␣
↪values for your application.
handel delete -c default-us-east-1 -e dev
```

When you execute that command, Handel will show you a big warning message like the following:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WARNING: YOU ARE ABOUT TO DELETE YOUR HANDEL ENVIRONMENT 'dev'!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

If you choose to delete this environment, you will lose all data stored in the␣
↪environment!

In particular, you will lose all data in the following:

* Databases
* Caches
* S3 Buckets
* EFS Mounts

PLEASE REVIEW this environment thoroughly, as you are responsible for all data loss␣
↪associated with an accidental deletion.
PLEASE BACKUP your data sources before deleting this environment just to be safe.
```

(continues on next page)

(continued from previous page)

```
? Enter 'yes' to delete your environment. Handel will refuse to delete the␣
↪environment with any other answer:
```

Type *yes* at the prompt to delete the environment. Handel will then proceed to delete the environment.

CHAPTER 14

Using Extensions

Handel provides an API for writing extensions to provide additional service types other than the official service types provided by Handel. Organizations can use this to implement service types that are highly customized to their particular use cases. These custom service types can retain the same ease-of-configuration and automatic service wiring that Handel provides.

> **Danger: Extensions are inherently dangerous!**
>
> Handel needs to run with administrator permissions, so **extensions can potentially harm your account** in many ways. Handel cannot validate what an extension is doing, so by using an extension you are running untrusted code.
>
> **DO NOT** run an extension unless you trust the source and have validated what actions it performs.

14.1 Using an Extension

Once you've found an extension that you want to use, you'll need to specify the extension to be loaded in your Handel file. You can then use the service types that extension provides.

In this section, we'll use the sns-handel-extension as an example. Handel already ships with an SNS service type, so this extension is really only useful as an example of how to consume extensions.

14.1.1 Load the Extension

To use an extension, first configure it to be loaded in your Handel file:

```
version: 1

name: sns-ext-example

extensions:
  sns: sns-handel-extension
```

The *extensions* section contains an object of one or more extensions you want Handel to load when you execute the project. The key is a short name that you can choose. You will use this short name when referencing the extension's service types. The value is the name of the NPM package containing the Handel extension.

Important: Since extensions are defined in your Handel file, that means they will only be loaded for that project and not globally for all projects.

If you have another project that is using Handel, you can use the same extension by configuring the *extensions* section in that Handel file to load the extension as well.

14.1.2 Use Extension Service Types

Once you have loaded the extensions that you'll be using, you can reference the service types contained in them:

```
version: 1

name: sns-ext-example

extensions:
  sns: sns-handel-extension

environments:
  dev:
    task:
      type: sns::sns
```

Note from the example above that when using extension services you must use the syntax *<extensionName>::<serviceType>*. In the above case we named our extension *sns* and the service type we are using in that extension is also called *sns*, which is why the resulting type you specify is *sns::sns*

Note: You can know what service types an extension contains, as well as how to configure each service type, by looking at the documentation provided by the extension.

14.2 Specifying an Extension Version

By default, Handel will grab the latest version of the specified extension from NPM. If you wish to specify a version or range of versions, you can use the syntax from the package.json spec:

```
version: 1

name: sns-ext-example

extensions:
  sns: sns-handel-extension@^0.1.0

environments:
  dev:
    task:
      type: sns::sns
```

This will cause Handel to fetch the latest 0.1.x version of the sns-handel-extension. For more about how these rules work, see the documentation on NPM's implementation of semantic versioning

14.3 Local Extensions

You may find yourself wanting to implement something that Handel doesn't support, but isn't widely reusable. While it is usually best to contribute an extension to the wider Handel ecosystem, there are cases where that is not appropriate.

Handel leverages NPM's support for local paths allows you to create 'local extensions' - extensions which live inside of your project.

You'll need to follow the guide to *Writing Extensions*, and put your extension source code in a subdirectory of your project: we recommend inside of a directory called *.local-handel-extensions*, but you can name it anything you like.

Let's say you've implemented an extension in *.local-handel-extensions/fancy-extension*. You can now use it like this:

```
version: 1

name: local-extension-example

extensions:
  fancy: file:.local-handel-extensions/fancy-extension

environments:
  dev:
    fancy:
      type: fancy:superfancy
```

Note: Handel will ensure that all production dependencies listed in your local extension's *package.json* are installed, but will not perform any build steps for you (like transpiling from Typescript).

You will need to ensure that any such build steps are carried out before running *handel*.

14.4 Other Extension Sources

Handel also supports installing extensions from GitHub, GitLab, Bitbucket, and Git repositories.

The values for these sources must be prefixed by their type ("*github:*", "*gitlab:*", "*bitbucket:*", "*git:*") and follow the format specified in the npm install documentation.

```
version: 1

name: local-extension-example

extensions:
  my-github-extension:     github:myorg/myrepo#my-optional-branch-specifier
  my-bitbucket-extension:  bitbucket:myuser/myrepo
  my-gitlab-extension:     gitlab:myorg/myrepo
  my-git-extension:        git:git+https://my-server.com/my-repo.git
```

CHAPTER 15

Alexa Skill Kit

This document contains information about the Alexa Skill kit service supported in Handel. This Handel service provisions a Alexa Skill kit permission, which is used to integrate with Lambda to invoke them.

Note: This service does not currently support resource tagging.

15.1 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *alexaskillkit* for this service type.

15.2 Example Handel Files

15.2.1 Example Lambda Config

This Handel file shows a Alexa Skill kit service being configured, producing to a Lambda:

```
version: 1

name: my-alexaskill-lambda

environments:
  dev:
    function:
      type: lambda
      path_to_code: .
      handler: app.handler
```

(continues on next page)

(continued from previous page)

```
    runtime: nodejs6.10
  alexaskill:
    type: alexaskillkit
    event_consumers:
    - service_name: function
```

15.3 Depending on this service

The Alexa Skill Kit service cannot be referenced as a dependency for another Handel service. This service is intended to be used as a producer of events for other services.

15.4 Events produced by this service

The Alexa Skill Kit service currently produces events for the following service types:

- Lambda

15.5 Events consumed by this service

The Alexa Skill Kit service does not consume events from other Handel services.

CHAPTER 16

AI Services

This document contains information about the AI Services provisioner supported in Handel. This Handel service allows you to access to services such as Rekognition in your application.

This service does not create any AWS resources since the AI services are consumed via an HTTP API. Even though you don't have provisioned resources, you still pay for each API call made to the AWS AI services.

16.1 Service Limitations

16.1.1 No Rekognition Streams Support

This service doesn't support Rekognition's Kinesis video stream processors.

16.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *aiservices* for this service type.
ai_services	list<string>	Yes		A list of one or more AWS AI services for which to add permissions. See Supported Service Access below for the list of services you can specify.

16.2.1 Supported Service Access

The following AWS services are supported in the *aws_services* element:

- *rekognition*
- *polly*

- *comprehend*
- translate
- transcribe

16.3 Rekognition

16.3.1 Collection Restrictions

Rekognition calls can be broken up into two general categories:

- Those dealing with individual images
- Those dealing with collections of persisted images

The individual image operations are stateless: In order to get the same results you must have the same image. The **image collections are NOT stateless; they persist information** about images you have added to the collection previously. For example, if you create a collection and add an image to it, the faces from that image will be indexed. Future calls to the collection will be able to derive information about individuals from the stored information in the collection.

Because of this, Handel restricts your use of collections to those named with a particular prefix:

```
<appName>-<environmentName>
```

You may create, modify, and delete collections for any collections whose name starts with the above prefix. You may not use any other collections outside this namespace. This helps prevent other applcations in the same AWS account from accessing collections to which they are not authorized.

If you want to use objects from a S3 bucket, see *S3 Object Access*

16.4 Polly

Polly calls can be generated from text files to form audio files. Each language has multiple voices to choose from, which can be specified in your configuration.

With 3000 or less characters, you can listen, download, or save immediately. For up to 100,000 characters your task must be saved to an S3 bucket.

Polly also restricts lexicon use to those with a particular prefix:

```
<appName>-<environmentName>
```

If you want to use objects from a S3 bucket, see *S3 Object Access*

16.5 Comprehend

AWS Comprehend examines text to perform a variety of functions. It can detect the dominant language of a document, entities, key phrases, sentiments (if a document is posative, negative, neutral, or mixed), syntax, and topic modeling.

There are no restrictions on the comprehend service.

If you want to use objects from a S3 bucket, see *S3 Object Access*

16.6 Translate

Amazon Translate translates documents from the following twelve languages in to english, and from English into these languages:

Arabic

Chinese (Simplifed)

Chinese (Traditional)

Czech

French

Greman

Italian

Japanese

Potuguese

Russian

Spanish

Turkish

AWS Translate does not currently have support for S3 or file uploads

16.7 Transcribe

AWS Transcribe recognizes speech in audio files, and turns that into text. It pulls an audio file from a S3 bucket, and thus you will need *S3 Object Access*. The output text file will be stored in the same S3 bucket. When these are delivered, they may contain customer content.

A file must be in one of the following formats:

MP3 Mp4 FLAC WAV

Your file also must be less than two hours in length. For the best results, use FLAC or WAV.

16.8 S3 Object Access

If you want to use objects from S3 rather than passing in bytes directly to the API calls, you must make sure your caller has permissions to the bucket.

Important: Rekognition will use the permissions from the role of the *caller*, so your application will need to have permissions to the S3 bucket it is telling Rekognition to look in.

Here is an example Handel file showing what is required to make this happen:

```
version: 1

name: my-apigateway-app

```

(continues on next page)

(continued from previous page)

```
environments:
  dev:
    app:
      type: apigateway
      path_to_code: .
      lambda_runtime: nodejs6.10
      handler_function: index.handler
      dependencies:
      - aiaccess
      - bucket # This is the important part
    aiaccess:
      type: aiservices
      ai_services:
      - rekognition
    bucket:
      type: s3
```

Notice that your API Gateway service in the above example needs to have a dependency on the *bucket* service. It can then tell Rekognition to look at objects in that bucket, because it has access to the bucket.

16.9 Depending on this service

You can reference this service as a dependency in other services. It does not export any environment variables. Instead, it will just add a policy on the dependent service to allow access to the services you listed.

16.10 Events produced by this service

The AI Services provisioner does not produce events for other Handel services to consume.

16.11 Events consumed by this service

The AI Services provisioner does not consume events from other Handel services.

CHAPTER 17

Amazon MQ

This document contains information about the Amazon MQ provisioner supported in Handel. This Handel service allows you to provision an ActiveMQ broker in AWS.

Warning: This provisioner is new and should be considered in beta. It is subject to breaking changes until this beta label is removed.

17.1 Service Limitations

17.1.1 No Custom Configuration Support

This service doesn't support providing a custom ActiveMQ configuration yet.

17.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *amazonmq* for this service type.
instance_type	string	No	mq.t2.micro	The Amazon MQ EC2 instance type that you wish to use for your broker. See Amazon MQ Pricing for details on the allowed instance types.
multi_az	boolean	No	false	Whether or not you want to deploy your broker in multi-AZ high availability mode.
general_logging	boolean	No	false	Whether or not you want general logging to be enabled for your broker.
audit_logging	boolean	No	false	Whether or not you want audit logging to be enabled for your broker.

17.3 Depending on this service

The Amazon MQ service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_BROKER_ID	The ID of the created broker.

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

17.4 Events produced by this service

The Amazon MQ provisioner does not produce AWS events for other Handel services to consume.

17.5 Events consumed by this service

The Amazon MQ provisioner does not consume AWS events from other Handel services.

CHAPTER 18

API Access

This document contains information about the API Access service supported in Handel. This Handel service allows you to add read-only access to AWS services in your application.

This service does not provision any AWS resources, it just serves to add additional permissions onto your applications.

Note: This service won't grant you permissions to publish to topics, read from data stores, etc. The permissions this service grants are read-only on the service level.

As an example of how you would use this service, you may want to run a Lambda that inspects your EC2 instances to audit them for certain characteristics. You can use this *apiaccess* service to grant that read-only access to EC2 to give you that information.

Since this service provides limited read-only access, in the EC2 example you would not be able to do things like start instances, create AMIs, etc.

Note: This service does not currently support resource tagging.

18.1 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *apiaccess* for this service type.
aws_services	List<string>	Yes		A list of one or more AWS services for which to add permissions. See Supported Service Access below for the list of services you can specify.

18.1.1 Supported Service Access

The following AWS services are supported in the *aws_services* element:

- beanstalk
- cloudformation
- cloudwatchevents
- codebuild
- codepipeline
- dynamodb
- ec2
- ecs
- efs
- elasticache
- lambda
- loadbalancing
- organizations
- rds
- route53
- s3
- sns
- sqs
- ssm

18.2 Example Handel File

This Handel file shows an API Gateway service being configured with API access to the Organizations service

```
version: 1

name: my-apigateway-app

environments:
  dev:
    app:
      type: apigateway
      path_to_code: .
      lambda_runtime: nodejs6.10
      handler_function: index.handler
    orgsaccess:
      type: apiaccess
      aws_services:
      - organizations
```

18.3 Depending on this service

You can reference this service as a dependency in other services. It does not export any environment variables. Instead, it will just add a policy on the dependent service to allow read access to the services you listed.

18.4 Events produced by this service

The API Access service does not produce events for other Handel services to consume.

18.5 Events consumed by this service

The API Access service does not consume events from other Handel services.

CHAPTER 19

API Gateway

This document contains information about the API Gateway service supported in Handel. This Handel service provisions resources such as API Gateway and Lambda to provide a serverless HTTP application.

19.1 Service Limitations

19.1.1 No Authorizer Lambdas

This service doesn't yet support specifying authorizer lambdas.

19.1.2 No Regional Endpoints

This service currently supports only edge-optimized API Gateways.

19.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *apigateway* for this service type.
proxy	*Proxy Passthrough*	No		Specify this section if you want a simple *proxy passthrough*, where all routes are directed to the same Lambda. You must specify either the *swagger* or *proxy* section, but not both.
swagger	*Swagger Configuration*	No		Specify this section if you want to configure your API from a Swagger document. You must specify either the *swagger* or *proxy* section, but not both.
description	string	No	Handel-created API	The configuration description of your Lambda function.
binary_media_types	array	No		A sequence (array) of BinaryMediaType strings. *Note* The handel will do the '/' to '~1' character escaping for you.
vpc	boolean	No	false	If true, your Lambdas will be deployed inside your account's VPC.
custom_domains	Array of *Custom Domain Mappings*	No		An array of custom domains to map to this API Gateway instance.
tags	*Resource Tags*	No		Any tags you want to apply to your API Gateway app.

19.2.1 Custom Domain Mappings

Note: This service does not currently support sharing custom domains between API Gateway instances using Base Path Mappings. At this time, you can only map one API Gateway to one custom domain, with no path mapping.

API Gateway allows for mapping gateways to one or more custom domains. These custom domains are always served via HTTPS.

The Custom Domains section is defined by the following schema:

```
custom_domains:
- dns_name: <string> # The DNS name for the API Gateway. Must be a valid DNS name.
  https_certificate: <arn> # The Amazon Certificate Manager certificate to use. This␣
↪certificate must be in the us-east-1 region.
```

See *DNS Records* for more information on how DNS records will be created.

19.2.2 Lambda Warmup

One of the challenges with servicing API requests with AWS Lambda is cold start times. Luckily, there are well-established patterns for reducing the impact of cold starts. One of these is to use CloudWatch scheduled events to make sure that there is always at least one instance of a lambda function warm and ready to service requests.

While the way one configures the warmup settings varies between Proxy and Swagger-based configuration, the basics are the same.

Parameters

Parameter	Type	Required	Default	Description
schedule	string	Yes		How often to send a warm up event. Either a rate or a cron expression. See CloudWatch for valid values.
http_paths	Array of strings	No		Can contain up to 5 entries. If specified, instead of sending a CloudWatch Event body, a simulated API Gateway *GET* event will be dispatched to each the provided paths.

By default, this will send a Cloudwatch scheduled event to the Lambda on the specified schedule. The event body looks like this:

```
{
  "version": "0",
  "id": "53dc4d37-cffa-4f76-80c9-8b7d4a4d2eaa",
  "detail-type": "Scheduled Event",
  "source": "aws.events",
  "account": "123456789012",
  "time": "2015-10-08T16:53:06Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:events:us-east-1:123456789012:rule/my-scheduled-rule"
  ],
  "detail": {}
}
```

In many cases, it is easier to simulate an API Gateway Proxy event instead of forcing the function to be able to consume multiple types of events. If you set one or more values for the *http_paths* array, instead of sending the default scheduled event body, an API Gateway Proxy event will be sent for each listed path. A maximum of 5 different paths may be specified for each function. These requests will always be GET requests and will not include any form of authentication information or custom headers. They will include a header called 'X-Lambda-Warmup' with a value matching the specified path.

19.2.3 Proxy Passthrough

Note: If you specify the *proxy* section, you may not specify the *swagger* section.

You specify the *proxy* section when you want a single Lambda function that handles all requests from all paths. Use this option when you only have a single route, or you want to handle routing in your code via a library.

The Proxy Passthrough section is defined by the following schema:

```
proxy:
  path_to_code: <string> # The path to the directory or artifact where your code␣
↪resides.
  runtime: <string> # The `Lambda runtime <http://docs.aws.amazon.com/lambda/latest/
↪dg/API_CreateFunction.html#SSS-CreateFunction-request-Runtime>`_ to use for your␣
↪handler function.
  handler: <string> # The function to call (such as index.handler) in your deployable␣
↪code when invoking the Lambda. This is the Lambda-equivalent of your 'main' method.
```

(continues on next page)

(continued from previous page)

```
  memory: <number> # The amount of memory (in MB) to provision for the runtime.␣
↪Default: 128
  timeout: <number> # The timeout to use for your Lambda function. Any functions that␣
↪go over this timeout will be killed. Default: 5
  warmup: # Optional. :ref:`apigateway-lambda-warmup`
    schedule: rate(5 minutes)
    http_paths:
    - /ping
  environment_variables: # A set of key/value pairs to set as environment variables␣
↪on your API.
    <STRING>: <string>
```

19.2.4 Swagger Configuration

Note: If you specify the *swagger* section, you may not specify the *proxy* section.

You specify the *swagger* section when you want to have your API defined by a Swagger document that is serviced by one or more Lambda functions in any combination.

The Swagger section is defined by the following schema:

```
swagger: <string> # The path to the Swagger file in your repository
```

19.2.5 Lambda Swagger Extensions

For the most part, the Swagger document you provide in the *swagger* section is just a regular Swagger document, specifying the API paths you want your app to use. If you're using Lambdas to service your API Gateway resources, Handel makes use of certain Swagger extensions in your Swagger document so that it can create and wire your Lambdas for you.

Consider the following Swagger document:

```
{
  "swagger": "2.0",
  "info": {
    "title": "my-cool-app",
    "description": "Test Swagger API",
    "version:": "1.0"
  },
  "paths": {
    "/": {
      "get": {
        "responses": {
          "200": {}
        },
        "x-lambda-function": "my-function-1"
      }
    }
  },
  "x-lambda-functions": {
    "my-function-1": {
      "runtime": "nodejs6.10",
```

(continues on next page)

(continued from previous page)

```
      "handler": "index.handler",
      "memory": "128",
      "path_to_code": "./function1"
      "warmup": {
        "schedule": "rate(5 minutes)",
        "http_paths": ["/"]
      }
    }
  }
}
```

Notice that this is just a vanilla Swagger document for the most part. It does have some Handel-provided extensions, however. Notice that the Swagger document contains an *x-lambda-functions* section. This section contains a list of elements that define Lambda configurations. For each item in this list, Handel will create a Lambda function for you. These objects are defined by the following schema:

```
{
  "path_to_code": <string>, // The path to the directory or artifact where your code␣
↪resides.
  "runtime": <string>, // The Lambda runtime (such as nodejs6.10) to use for your␣
↪handler function.
  "handler": <string>, // The function to call (such as index.handler) in your␣
↪deployable code when invoking the Lambda. This is the Lambda-equivalent of your␣
↪'main' method.
  "memory": <number>, // The amount of memory (in MB) to provision for the runtime.␣
↪Default: 128,
  "timeout": <number>, // The timeout to use for your Lambda function. Any functions␣
↪that go over this timeout will be killed. Default: 5
  "warmup": { // The :ref:`apigateway-lambda-warmup` configuration
    "schedule": "rate(5 minutes)",
    "http_paths": [
      "<string>" // Path relative to the API Gateway root to invoke on the schedule.
    ]
  },
  "environment_variables": { // A set of key/value pairs to set as environment␣
↪variables on your API.
    <ENV_NAME>: <env value>
  }
}
```

Also notice that the paths in your document have an *x-lambda-function* element. This element tells Handel which Lambda function from the *x-lambda-functions* section you want that API path to be serviced by.

The above example just shows the easy Lambda proxy functionality in API Gateway. This will effectively pass all requests through to your Lambda without modification. If you want to use API Gateway's integration functionality to have more complex transformations before sending requests to your Lambda, you can use Handel to do this. Just provide the regular Amazon *x-amazon-apigateway-integration* value in your Swagger file:

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-09-12T23:19:28Z",
    "title": "MyAPI"
  },
  "basePath": "/test",
  "schemes": [
```

(continues on next page)

(continued from previous page)

```
    "https"
  ],
  "paths": {
    "/{myparam}": {
      "get": {
        "produces": [
          "application/json"
        ],
        "responses": {},
        "x-lambda-function": "my-function-1"
        "x-amazon-apigateway-integration": {
          "requestTemplates": {
            "application/json": "#set ($root=$input.path('$')) { \"stage\": \"$root.
↪name\", \"user-id\": \"$root.key\" }",
            "application/xml": "#set ($root=$input.path('$')) <stage>$root.name</
↪stage> "
          },
          "requestParameters": {
            "integration.request.path.myparam": "method.request.querystring.version",
            "integration.request.querystring.provider": "method.request.querystring.
↪vendor"
          },
          "cacheNamespace": "cache namespace",
          "cacheKeyParameters": [],
          "responses": {
            "2\\d{2}": {
              "statusCode": "200",
              "responseParameters": {
                "method.response.header.requestId": "integration.response.header.cid"
              },
              "responseTemplates": {
                "application/json": "#set ($root=$input.path('$')) { \"stage\": \"
↪$root.name\", \"user-id\": \"$root.key\" }",
                "application/xml": "#set ($root=$input.path('$')) <stage>$root.name</
↪stage> "
              }
            },
            "302": {
              "statusCode": "302",
              "responseParameters": {
                "method.response.header.Location": "integration.response.body.
↪redirect.url"
              }
            },
            "default": {
              "statusCode": "400",
              "responseParameters": {
                "method.response.header.test-method-response-header": "'static value'"
              }
            }
          }
        }
      }
    }
  }
  "x-lambda-functions": {
    "my-function-1": {
```

(continues on next page)

(continued from previous page)

```
      "runtime": "nodejs6.10",
      "handler": "index.handler",
      "memory": "128",
      "path_to_code": "./function1"
    }
  }
}
```

Notice that the above example has omitted the Lambda-specific properties in the integration object, such as *uri*. Handel will still create and wire the Lambdas for you.

19.2.6 HTTP Passthrough Swagger Extensions

In addition to servicing your API methods with Lambdas, you can configure API Gateway to just do an HTTP passthrough to some other HTTP endpoint, be it an AWS EC2 server or something else outside of AWS entirely.

Handel supports this with another swagger extension, called *x-http-passthrough-url* that you configure on your resource methods. Here's an example:

```
{
  "swagger": "2.0",
  "info": {
    "title": "my-cool-app",
    "description": "Test Swagger API",
    "version:": "1.0"
  },
  "paths": {
    "/": {
      "get": {
        "responses": {
          "200": {}
        },
        "x-http-passthrough-url": "https://my.cool.fake.url.com"
      }
    }
  }
}
```

The above Swagger document will route GET on the "/" path to "https://my.cool.fake.url.com". All request headers, parameters, and body will be passed through directly to the given URL, and the response from the URL will be passed through API Gateway without modification.

If you need to use path params with the HTTP passthrough, you can use the *x-http-passthrough-path-params* Swagger extension to map the path parameters from the API Gateway request to the HTTP backend request. Here's an example Swagger document doing this:

```
{
  "swagger": "2.0",
  "info": {
    "title": "my-cool-app",
    "description": "Test Swagger API",
    "version:": "1.0"
  },
  "paths": {
    "/user/{name}": {
```

(continues on next page)

(continued from previous page)

```
      "get": {
        "responses": {
          "200": {}
        },
        "x-http-passthrough-url": "https://my.cool.fake.url.com/{person}",
        "x-http-passthrough-path-params": {
          "name": "person"
        }
      }
    }
  }
}
```

The above example shows mapping the "name" path parameter in the API Gateway request to the "person" path parameter in the backend request.

19.3 Example Handel File

19.3.1 Simple Proxy Passthrough

This Handel file shows an API Gateway service being configured, where all your requests on all paths go to a single Lambda function:

```
version: 1

name: my-apigateway-app

environments:
  dev:
    app:
      type: apigateway
      proxy:
        path_to_code: .
        runtime: nodejs6.10
        handler: index.handler
        memory: 256
        timeout: 5
        environment_variables:
          MY_FIRST_VAR: my_first_value
          MY_SECOND_VAR: my_second_value
```

19.3.2 Swagger Configuration

This Handel file shows an API Gateway service being configured, where your API definition is defined by a Swagger file:

```
version: 1

name: my-apigateway-app

environments:
  dev:
```

(continues on next page)

(continued from previous page)

```
    app:
      type: apigateway
      swagger: ./swagger.json
```

The above file assumes a Swagger file called *swagger.json* is present in the same directory as the Handel file. Here is an example Swagger file:

```
{
  "swagger": "2.0",
  "info": {
    "title": "my-cool-app",
    "description": "Test Swagger API",
    "version:": "1.0"
  },
  "paths": {
    "/": {
      "get": {
        "responses": {
          "200": {}
        },
        "x-lambda-function": "my-function-1"
      }
    },
    "/test1": {
      "get": {
        "responses": {
          "200": {}
        },
        "x-lambda-function": "my-function-2"
      }
    }
  },
  "x-lambda-functions": {
    "my-function-1": {
      "runtime": "nodejs6.10",
      "handler": "index.handler",
      "memory": "128",
      "path_to_code": "./function1"
    },
    "my-function-2": {
      "runtime": "nodejs6.10",
      "handler": "index.handler",
      "memory": "256",
      "path_to_code": "./function2"
    }
  }
}
```

19.4 Depending on this service

The API Gateway service cannot be referenced as a dependency for another Handel service

19.5 Events produced by this service

The API Gateway service does not produce events for other Handel services to consume.

19.6 Events consumed by this service

The API Gateway service does not consume events from other Handel services.

CHAPTER 20

Aurora (RDS)

This page contains information about using the Aurora service in Handel. This service provides an Aurora cluster (MySQL or PostgreSQL) via the RDS service.

20.1 Service Limitations

20.1.1 No Option Group Support

This service doesn't allow you to specify any custom options in an option group. It does allow you specify custom parameters in a parameter group, however.

20.1.2 No Update Support

This service intentionally does not support updates. Once a database is created, certain updates to the database will cause a new database to be created and the old one deleted. In an effort to avoid unwanted data loss, we don't update this service automatically. You can still modify the database and parameter group manually in the AWS console.

Warning: Make sure you know what you're doing when you modify your RDS database in the AWS Console. Certain actions will cause database downtime, and some may even cause the database to be recreated.

20.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *aurora* for this service type.
engine	string	Yes		The Aurora engine you wish to use. Allowed values: 'mysql', 'postgresql'
version	string	Yes		The version of MySQL or PostgreSQL you wish to run. Allowed values for MySQL: '5.7.12'. Allowed values for PostgreSQL: '9.6.3'.
database_name	string	Yes		The name of your database in your Aurora cluster.
description	string	No		The description on the resources created for the cluster
instance_type	string	No	db.t2.small for MySQL, db.r4.large for PostgreSQL.	The size of database instance to run. Not all database instance types are supported for Aurora.
cluster_size	number	No	1	The number of instances (including the primary) to run in your cluster.
cluster_parameters	map<string, string>	No		A list of key/value Aurora cluster parameter group pairs to configure your cluster. You will need to look in the AWS Console to see the list of available cluster parameters for Aurora.
instance_parameters	map<string, string>	No		A list of key/value Aurora instance parameter group pairs to configure the instances in your cluster. You will need to look in the AWS Console to see the list of available instance parameters for Aurora.
tags	*Resource Tags*	No		Any tags you wish to apply to this Aurora instance.

Warning: Be aware that Aurora clusters can be very expensive. A cluster with 3 *db.r4.2xlarge* instances in it will cost about about $2,500/month. Make sure you check how much you will be paying!

You can use the excellent EC2Instances.info site to easily see pricing information for RDS databases. Remember that you pay the full price for each instance in your cluster.

20.3 Example Handel File

```
version: 1

name: aurora-test

environments:
  dev:
    database:
        type: aurora
        engine: mysql
        version: 5.7.12
```

(continues on next page)

(continued from previous page)

```
        database_name: MyDb
        instance_type: db.t2.medium
        cluster_size: 3
        cluster_parameters: # This is where you can set parameters that configure the␣
↪cluster as a whole
            character_set_database: utf8mb4
        instance_parameters: # This is where you can set parameters that apply to␣
↪each instance.
            autocommit: 1
        tags:
            some: tag
```

20.4 Depending on this service

The Aurora service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_CLUSTER_ENDPOINT	The address that you should use for writes to the database.
<SERVICE_NAME>_READ_ENDPOINT	The address that you should use for reads to the database.
<SERVICE_NAME>_PORT	The port on which the Aurora cluster instances are listening.
<SERVICE_NAME>_DATABASE_NAME	The name of the database in your Aurora cluster.

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

In addition, the Aurora service puts the following credentials into the EC2 parameter store:

Parameter Name	Description
<parameter_prefix>.<service_name>.db_username	The username for your database user.
<parameter_prefix>.<service_name>.db_password	The password for your database user.

Note: The <parameter_prefix> section of the parameter name is a consistent prefix applied to all parameters injected by services in the EC2 Parameter Store. See *Parameter Store Prefix* for information about the structure of this prefix.

The <service_name> section of the parameter name should be replaced by the *service name* you gave your database in your Handel file.

20.5 Events produced by this service

The Aurora service does not produce events for other Handel services to consume.

20.6 Events consumed by this service

The Aurora service does not consume events from other Handel services.

CHAPTER 21

Aurora Serverless

This page contains information about using the Aurora Serverless service in Handel. This service provides a "serverless" instance of Aurora (MySQL).

Warning: Aurora Serverless is not appropriate for all workloads. Review the Use Cases before choosing this service.

21.1 Service Limitations

21.1.1 No Option Group Support

This service doesn't allow you to specify any custom options in an option group. It does allow you specify custom parameters in a parameter group, however.

21.1.2 No Update Support

This service intentionally does not support updates. Once a database is created, certain updates to the database will cause a new database to be created and the old one deleted. In an effort to avoid unwanted data loss, we don't update this service automatically. You can still modify the database and parameter group manually in the AWS console.

Warning: Make sure you know what you're doing when you modify your RDS database in the AWS Console. Certain actions will cause database downtime, and some may even cause the database to be recreated.

21.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *aurora-serverless* for this service type.
engine	string	Yes		The Aurora engine you wish to use. Allowed values: 'mysql'
version	string	Yes		The version of MySQL you wish to run. Allowed values for MySQL: '5.6.10a'
database_name	string	Yes		The name of your database in your Aurora cluster.
description	string	No		The description on the resources created for the cluster
scaling	*Scaling Configuration*	No		Cluster capacity scaling configuration
cluster_parameters	map<string, string>	No		A list of key/value Aurora cluster parameter group pairs to configure your cluster. You will need to look in the AWS Console to see the list of available cluster parameters for Aurora.
tags	*Resource Tags*	No		Any tags you wish to apply to this Aurora instance.

21.2.1 Scaling Configuration

The *scaling* section is defined by the following schema:

Parameter	Type	Required	Default	Description
auto_pause	boolean	No	true	Whether to automatically pause this database if it has been idle for a specified time.
seconds_until_auto_pause	number	No	*300* (5 minutes)	How long the database must be idle before it can be paused.
min_capacity	One of *2*, *4*, *8*, *16*, *32*, *64*, *128*, or *256*	No	*2*	The minimum capacity (in Aurora Compute Units)
max_capacity	One of *2*, *4*, *8*, *16*, *32*, *64*, *128*, or *256*	No	*64*	The maximum capacity (in Aurora Compute Units)

21.3 Example Handel File

```
version: 1

name: aurora-serverless-test

environments:
  dev:
    database:
        type: aurora-serverless
        engine: mysql
        version: 5.6.10a
```

(continues on next page)

(continued from previous page)

```
        database_name: MyDb
        scaling:
            min_capacity: 2
            max_capacity: 16
            auto_pause: true
            seconds_until_auto_pause: 600 # 10 minutes
        cluster_parameters: # This is where you can set parameters that configure the␣
↪cluster as a whole
            character_set_database: utf8mb4
        tags:
            some: tag
```

21.4 Depending on this service

The Aurora Serverless service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_CLUSTER_ENDPOINT	The address that you should use for writes to the database.
<SERVICE_NAME>_READ_ENDPOINT	The address that you should use for reads to the database.
<SERVICE_NAME>_PORT	The port on which the Aurora cluster instances are listening.
<SERVICE_NAME>_DATABASE_NAME	The name of the database in your Aurora cluster.

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

In addition, the Aurora service puts the following credentials into the EC2 parameter store:

Parameter Name	Description
<parameter_prefix>.<service_name>.db_username	The username for your database user.
<parameter_prefix>.<service_name>.db_password	The password for your database user.

Note: The <parameter_prefix> section of the parameter name is a consistent prefix applied to all parameters injected by services in the EC2 Parameter Store. See *Parameter Store Prefix* for information about the structure of this prefix.

The <service_name> section of the parameter name should be replaced by the *service name* you gave your database in your Handel file.

Note: Aurora Serverless does not actually differentiate between read endpoints and write endpoints, like Aurora does. However, a common use case for Aurora Serverless is to run non-production workloads and to run the production workloads using provisioned Aurora. In order to make this use case simpler, the Aurora-Serverless Handel service mimics the variables set by the provisioned Aurora service.

21.5 Events produced by this service

The Aurora service does not produce events for other Handel services to consume.

21.6 Events consumed by this service

The Aurora service does not consume events from other Handel services.

CHAPTER 22

CodeDeploy

This document contains information about the CodeDeploy service supported in Handel. This Handel service provisions an autoscaling group running CodeDeploy. You can install arbitrary software on these instances using CodeDeploy's appspec.yml file.

Important: CodeDeploy is far less managed than other compute services like Lambda, ECS Fargate, and Elastic Beanstalk. You are responsible for all configuration on the EC2 instances. Please see the CodeDeploy Documentation for details on this service,

22.1 Service Limitations

22.1.1 No Windows Support

This service currently doesn't allow you to provision Windows instances to use with CodeDeploy.

22.1.2 No Single Instance Support

This service doesn't support using CodeDeploy in a single-instance configuration. It only supports using auto-scaling groups, although you can use an auto-scaling group with a min/max of 1, which gets you a single instance.

22.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *codedeploy* for this service type.
path_to_code	string	Yes		The location of the directory you want to upload to CodeDeploy. You must have your *appspec.yml* file at the root of this directory!
os	string	Yes		The type of OS to use with CodeDeploy. Currently the only supported value is *linux*.
instance_type	string	No	t2.micro	The EC2 instance type on which your application will run.
key_name	string	No	None	The name of the EC2 keypair to use for SSH access to the instances.
auto_scaling	*AutoScaling*	No		The configuration to use for scaling up and down
routing	*Routing*	No		The Routing element details what kind of routing you want to your CodeDeploy service (if any)
environment_variables	*EnvironmentVariables*	No		Any user-specified environment variables to inject in the application.
tags	*Resource Tags*	No		Any tags you want to apply to your CodeDeploy resources.

22.2.1 AutoScaling

The *auto_scaling* section is defined by the following schema:

```
auto_scaling: # Optional
  min_instances: <integer> # Optional. Default: 1
  max_instances: <integer> # Optional. Default: 1
  scaling_policies: # Optional
  - type: <up|down>
    adjustment:
      type: <string> # Optional. Default: 'ChangeInCapacity'.
      value: <number> # Required
      cooldown: <number> # Optional. Default: 300.
    alarm:
      namespace: <string> # Optional. Default: 'AWS/EC2'
      dimensions: # Optional. Default: Your auto-scaling group dimensions.
        <string>: <string>
      metric_name: <string> # Required
      statistic: <string> # Optional. Default: 'Average'
      comparison_operator: <string> # Required
      threshold: <number> # Required
      period: <number> # Optional. Default: 300
      evaluation_periods: <number> # Optional. Default: 5
```

Tip: Auto-scaling in AWS is based off the CloudWatch service. Configuring auto-scaling can be a bit daunting at first if you haven't used CloudWatch metrics or alarms.

See the below *Example Handel Files* section for some examples of configuring auto-scaling.

22.2.2 EnvironmentVariables

The EnvironmentVariables element is defined by the following schema:

```
environment_variables:
  <YOUR_ENV_NAME>: <your_env_value>
```

<YOUR_ENV_NAME> is a string that will be the name of the injected environment variable. <your_env_value> is its value. You may specify an arbitrary number of environment variables in this section.

22.2.3 Routing

The Routing element is defined by the following schema:

```
routing:
  type: <http|https>
  https_certificate: <string> # Required if you select https as the routing type
  dns_names:
   - <string> # Optional
```

The *dns_names* section creates one or more dns names that point to this load balancer. See *DNS Records* for more.

22.3 Example Handel Files

22.3.1 Simple CodeDeploy Service

This Handel file shows the simplest possible CodeDeploy service. It doesn't have a load balancer to route requests to it, and it doesn't use auto-scaling.

```
version: 1

name: codedeploy-example

environments:
  dev:
    webapp:
      type: codedeploy
      path_to_code: .
      os: linux
```

22.3.2 CodeDeploy With Load Balancer

This Handel file shows a CodeDeploy service with a load balancer configured in front of it:

```
version: 1

name: codedeploy-example

environments:
  dev:
    webapp:
      type: codedeploy
```

(continues on next page)

(continued from previous page)

```
      path_to_code: .
      os: linux
      routing:
        type: https
        https_certificate: your-certificate-id-here
        dns_names: # Optional
        - mydnsname.myfakedomain.com
```

22.3.3 CodeDeploy With Auto-Scaling

This Handel file shows a CodeDeploy service with a load balancer and auto scaling policies configured:

```
version: 1

name: codedeploy-test

environments:
  dev:
    webapp:
      type: codedeploy
      path_to_code: .
      os: linux
      auto_scaling:
        min_instances: 1
        max_instances: 4
        scaling_policies:
        - type: up
          adjustment:
            value: 1
            cooldown: 60
          alarm:
            metric_name: CPUUtilization
            comparison_operator: GreaterThanThreshold
            threshold: 70
            period: 60
        - type: down
          adjustment:
            value: 1
            cooldown: 60
          alarm:
            metric_name: CPUUtilization
            comparison_operator: LessThanThreshold
            threshold: 30
            period: 60
      routing:
        type: https
        https_certificate: your-certificate-id-here
        dns_names:
        - mydnsname.myfakedomain.com
```

22.4 Depending on this service

The CodeDeploy service cannot be referenced as a dependency for another Handel service.

22.5 Events produced by this service

The CodeDeploy service does not produce events for other Handel services to consume.

22.6 Events consumed by this service

The CodeDeploy service does not consume events from other Handel services.

CHAPTER 23

Beanstalk

This document contains information about the Beanstalk service supported in Handel. This Handel service provisions an Elastic Beanstalk application, which consists of an auto-scaling group fronted by an Elastic Load Balancer.

23.1 Service Limitations

23.1.1 No WAR support

This Handel Beanstalk service does not yet support Java WAR stack types. Support is planned to be added in the near future.

23.1.2 Limited Tagging Support

Attention: CloudFormation doesn't allow Beanstalk tags to be modified after initial environment creation. Beanstalk just recently added support for updating tags, but CloudFormation doesn't yet support that feature change for Beanstalk.

Until this support is added, if you try to modify your *tags* element after your environment is created, your CloudFormation stack will fail to update.

23.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *beanstalk* for this service type.
path_to_code	string	Yes		The location of your code to upload to Beanstalk. This can be a directory (which will be zipped up) or a single file (such as a deployable Java WAR file). If this points to a directory containing a Dockerrun.aws.json file or points to a Dockerrun.aws.json file then the following *Dockerrun.aws.json Replacement Tags* will be substituted.
solution_stack	string	Yes		The ElasticBeanstalk solution stack you wish to use. This determines what AMI your application runs on. See Elastic Beanstalk Supported Platforms for the list of solution stacks.
description	string	No	Application.	The description of the application.
key_name	string	No	None	The name of the EC2 keypair to use for SSH access to the instance.
auto_scaling	*AutoScaling*	No		The configuration to use for scaling up and down
instance_type	string	No	t2.micro	The EC2 instance type on which your application will run.
health_check_url	string	No	/	The URL the ELB should use to check the health of your application.
routing	*Routing*	No		The Routing element details what kind of routing you want to your Beanstalk service
environment_variables	*Environment Variables*	No		Any user-specified environment variables to inject in the application.
tags	*Resource Tags*	No		Any tags you want to apply to your Beanstalk environment

23.3 Dockerrun.aws.json Replacement Tags

Tag	Description
<aws_account_id>	The account_id from the account config file specified at deployment.
<aws_region>	The region from the account config file specified at deployment.
<handel_app_name>	The name of the Handel application
<handel_environment_name>	The name of the Handel environment that the deployed service is contained in.
<handel_service_name>	The name of the Handel service being deployed.

23.3.1 AutoScaling

The *auto_scaling* section is defined by the following schema:

```
auto_scaling: # Optional
  min_instances: <integer> # Optional. Default: 1
  max_instances: <integer> # Optional. Default: 1
  scaling_policies: # Optional
  - type: <up|down>
    adjustment:
      type: <string> # Optional. Default: 'ChangeInCapacity'.
      value: <number> # Required
      cooldown: <number> # Optional. Default: 300.
    alarm:
      namespace: <string> # Optional. Default: 'AWS/EC2'
      dimensions: # Optional. Default: Your auto-scaling group dimensions.
        <string>: <string>
      metric_name: <string> # Required
      statistic: <string> # Optional. Default: 'Average'
      comparison_operator: <string> # Required
      threshold: <number> # Required
      period: <number> # Optional. Default: 300
      evaluation_periods: <number> # Optional. Default: 5
```

Tip: Auto-scaling in AWS is based off the CloudWatch service. Configuring auto-scaling can be a bit daunting at first if you haven't used CloudWatch metrics or alarms.

See the below *Example Handel Files* section for some examples of configuring auto-scaling.

23.3.2 EnvironmentVariables

The EnvironmentVariables element is defined by the following schema:

```
environment_variables:
  <YOUR_ENV_NAME>: <your_env_value>
```

<YOUR_ENV_NAME> is a string that will be the name of the injected environment variable. <your_env_value> is its value. You may specify an arbitrary number of environment variables in this section.

23.3.3 Routing

The Routing element is defined by the following schema:

```
routing:
  type: <http|https>
  https_certificate: <string> # Required if you select https as the routing type
  dns_names:
   - <string> # Optional
```

The *dns_names* section creates one or more dns names that point to this load balancer. See *DNS Records* for more.

23.4 Example Handel Files

23.4.1 Simple Beanstalk Service

This Handel file shows a simply-configured Beanstalk service with most of the defaults intact:

```
version: 1

name: my-beanstalk-app

environments:
  dev:
    webapp:
      type: beanstalk
      path_to_code: .
      solution_stack: 64bit Amazon Linux 2016.09 v4.0.1 running Node.js
      environment_variables:
        MY_INJECTED_VAR: myValue
```

23.4.2 Auto-Scaling On Service CPU Utilization

This Handel file shows a Beanstalk service auto-scaling on its own CPU Utilization metric. Note that in the *alarm* section you can leave off things like *namespace* and *dimensions* and it will default to your Beanstalk service for those values:

```
version: 1

name: beanstalk-example

environments:
  dev:
    webapp:
      type: beanstalk
      path_to_code: .
      solution_stack: 64bit Amazon Linux 2017.03 v4.1.0 running Node.js
      auto_scaling:
        min_instances: 1
        max_instances: 2
        scaling_policies:
        - type: up
          adjustment:
            value: 1
            cooldown: 60
          alarm:
            metric_name: CPUUtilization
            comparison_operator: GreaterThanThreshold
            threshold: 70
            period: 60
        - type: down
          adjustment:
            value: 1
            cooldown: 60
          alarm:
            metric_name: CPUUtilization
            comparison_operator: LessThanThreshold
```

(continues on next page)

(continued from previous page)

```
            threshold: 30
            period: 60
```

23.4.3 Auto-Scaling On Queue Size

This Handel file shows a Beanstalk service scaling off the size of a queue it consumes:

```
version: 1

name:  my-beanstalk-app

environments:
  dev:
    webapp:
      type: beanstalk
      path_to_code: .
      solution_stack: 64bit Amazon Linux 2017.03 v4.1.0 running Node.js
      auto_scaling:
        min_instances: 1
        max_instances: 2
        scaling_policies:
        - type: up
          adjustment:
            value: 1
          alarm:
            namespace: AWS/SQS
            dimensions:
              QueueName: my-beanstalk-app-dev-queue-sqs
            metric_name: ApproximateNumberOfMessagesVisible
            comparison_operator: GreaterThanThreshold
            threshold: 2000
        - type: down
          adjustment:
            value: 1
          alarm:
            namespace: AWS/SQS
            dimensions:
              QueueName: my-beanstalk-appe-dev-queue-sqs
            metric_name: ApproximateNumberOfMessagesVisible
            comparison_operator: LessThanThreshold
            threshold: 100
      dependencies:
      - queue
    queue:
      type: sqs
```

23.5 Depending on this service

The Beanstalk service cannot be referenced as a dependency for another Handel service.

23.6 Events produced by this service

The Beanstalk service does not produce events for other Handel services to consume.

23.7 Events consumed by this service

The Beanstalk service does not consume events from other Handel services.

CHAPTER 24

CloudWatch Events

This document contains information about the CloudWatch Events service supported in Handel. This Handel service provisions a CloudWatch Events rule, which can then be integrated with services like Lambda to invoke them when events fire.

Important: This service only offers limited tagging support. Cloudwatch events will not be tagged, but the Cloudformation stack used to create them will be. See *Tagging Unsupported Resources*.

24.1 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *cloudwatchevent* for this service type.
description	string	No	Handel-created rule.	The event description.
schedule	string	No		The cron or rate string specifying the schedule on which to fire the event. See the Scheduled Events document for information on the syntax of these schedule expressions.
event_pattern	object	No		The list of event patterns on which to fire the event. In this field you just specify an Event Pattern in YAML syntax.
state	string	No	enabled	What state the rule should be in. Allowed values: 'enabled', 'disabled'
tags	*Resource Tags*	No		Tags to be applied to the Cloudformation stack which provisions this resource.

24.2 Example Handel Files

24.2.1 Scheduled Lambda

This Handel file shows a CloudWatch Events service being configured, producing to a Lambda on a schedule:

```
version: 1

name: my-scheduled-lambda

environments:
  dev:
    function:
      type: lambda
      path_to_code: .
      handler: app.handler
      runtime: nodejs6.10
    schedule:
      type: cloudwatchevent
      schedule: rate(1 minute)
      event_consumers:
      - service_name: function
        event_input: '{"some": "param"}'
```

24.2.2 EBS Events Lambda

This Handel file shows a CloudWatch Events service being configured, producing to a Lambda when an EBS volume is created:

```
version: 1

name: my-event-lambda

environments:
  dev:
    function:
      type: lambda
      path_to_code: .
      handler: app.handler
      runtime: nodejs6.10
    schedule:
      type: cloudwatchevent
      event_pattern:
        source:
        - aws.ec2
        detail-type:
        - EBS Volume Notification
        detail:
        event:
        - createVolume
      event_consumers:
      - service_name: function
```

24.3 Depending on this service

The CloudWatch Events service cannot be referenced as a dependency for another Handel service. This service is intended to be used as a producer of events for other services.

24.4 Events produced by this service

The CloudWatch Events service currently produces events for the following services types:

- Lambda

24.5 Events consumed by this service

The CloudWatch Events service does not consume events from other Handel services.

CHAPTER 25

DynamoDB

This page contains information about using DynamoDB service supported in Handel. This service provisions a DynamoDB table for use by other AWS services.

25.1 Service Limitations

25.1.1 No Update Support

This service intentionally does not support updates. Once a table is created, certain updates to the table will cause a new one to be created and the old one deleted. In an effort to avoid unwanted data loss, we don't update this service automatically.

25.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *dynamodb* for this service type.
table_name	string	No	'<app name>-<environment>-<service name>-dynamodb'	Sets the name of the Dynamod table to be created.
partition_key	*PartitionKey*	Yes		The ParitionKey element details how you want your partition key specified.
sort_key	*SortKey*	No	None	The SortKey element details how you want your sort key specified. Unlike partition_key, sort_key is not required.
provisioned_throughput	*ProvisionedThroughput*	No	1 for read and write	The ProvisionedThroughput element details how much provisioned IOPS you want on your table for reads and writes.
ttl_attribute	string	No	None	Configures the attribute to use for DynamoDB TTL and auto-expiration.
local_indexes	*LocalIndexes*	No		You can configure local secondary indexes for fast queries on a different sort key within the same partition key.
stream_view_type	string	No		When present, the stream view type element indicates that a dynamodb stream will be used and specifies what information is written to the stream. Options are KEYS_ONLY, NEW_IMAGE, OLD_IMAGE and NEW_AND_OLD_IMAGES.
global_indexes	*GlobalIndexes*	No		You can configure global secondary indexes for fast queries on other partition and sort keys in addition to the ones on your table.
tags	*Resource Tags*	No		Any tags you want to apply to your Dynamo Table

25.2.1 PartitionKey

The PartitionKey element tells how to configure your partition key in DynamoDB. It has the following schema:

```
partition_key:
  name: <key_name>
  type: <String|Number>
```

25.2.2 SortKey

The SortKey element tells how to configure your sort key in DynamoDB. It has the following schema:

```
sort_key:
  name: <key_name>
  type: <String|Number>
```

25.2.3 ProvisionedThroughput

The ProvisionedThroughput element tells many IOPS to provision for your table for reads and writes. It has the following schema:

```
provisioned_throughput:
  read_capacity_units: <number or range> # Required
  write_capacity_units: <number or range> # Required
  read_target_utilization: <percentage> # Default: 70 (if autoscaling is enabled)
  write_target_utilization: <percentage> # Default: 70 (if autoscaling is enabled)
```

Autoscaling Throughput

If a range (ex: 1-10) is provided to *read_capacity_units* or *write_capacity_units*, an autoscaling rule will be created with the min and max values from the range and target utilization as specified by *read_target_utilization* and *write_target_utilization*.

The following configuration will cause the read capacity to be automatically scaled between 10 and 100, with a target usage of 50%. The write capacity will scale between 1-10, with a target usage of 70% (the default).

```
provisioned_throughput:
  read_capacity_units: 10-100
  write_capacity_units: 1-10
  read_target_utilization: 50
```

25.2.4 LocalIndexes

The LocalIndexes element allows you to configure local secondary indexes on your table for alternate query methods. It has the following schema:

```
local_indexes:
- name: <string> # Required
  sort_key: # Required
    name: <string>
    type: <String|Number>
  attributes_to_copy: # Required
  - <string>
```

25.2.5 GlobalIndexes

The GlobalIndexes element allows you to configure global secondary indexes on your table for alternate query methods. It allows you to specify a different partition key than the main table. It has the following schema:

```
global_indexes:
- name: <string> # Required
  partition_key: # Required
    name: <string>
    type: <String|Number>
  sort_key: # Optional
    name: <string>
    type: <String|Number>
  attributes_to_copy: # Optional. If not specified, will default to ALL
  - <string>
```

(continues on next page)

(continued from previous page)

```
  provisioned_throughput: # Optional
    read_capacity_units: <number or range> # Required
    write_capacity_units: <number or range> # Required
    read_target_utilization: <percentage> # Default: Matches table config
    write_target_utilization: <percentage> # Default: Matches table config
```

The provisioned throughput configuration for Global Secondary Indexes matches that for the table. If the provisioned throughput is not configured for the index, the table's configuration will be used, including any autoscaling configuration.

Warning: Be aware that using Global Secondary Indexes can greatly increase your cost. When you use global indexes, you are effectively creating a new table. This will increase your cost by the amount required for storage and allocated IOPS for the global index.

25.3 Example Handel File

```
version: 1

name: my-ecs-app

environments:
  dev:
    webapp:
      type: dynamodb
      partition_key: # Required, NOT updateable
        name: MyPartionKey
        type: String
      sort_key:
        name: MySortKey
        type: Number
      provisioned_throughput:
        read_capacity_units: 1-20 #Autoscale reads, but not writes
        write_capacity_units: 6
      tags:
        name: my-dynamodb-tag
```

25.4 Depending on this service

The DynamoDB service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_TABLE_NAME	The name of the created DynamoDB table
<SERVICE_NAME>_TABLE_ARN	The ARN of the created DynamoDB table

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

25.5 DynamoDB Streams

A DynamoDB Stream sends an event to a lambda function when data in the table changes. To configure a stream, include the stream_view_type element in your handel file and declare your lambda function as an event_consumer with the following syntax:

```
event_consumers:
- service_name: <string> # Required.  The service name of the lambda function
  batch_size: <number> # Optional.  Default: 100
```

25.5.1 BatchSize

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records. The default is 100 records.

25.6 Events produced by this service

The DynamoDB service currently produces events for the following services types:

- Lambda

25.7 Events consumed by this service

The DynamoDB service does not consume events from other Handel services.

CHAPTER 26

ECS (Elastic Container Service)

This page contains information about the ECS service supported in Handel. This Handel service provisions your application code as an ECS Service, with included supporting infrastructure such as load balancers and auto-scaling groups.

26.1 Service Limitations

26.1.1 One service per cluster

This service uses a model of one ECS service per ECS cluster. It does not support the model of one large cluster with multiple services running on it.

26.1.2 Unsupported ECS task features

This service currently does not support the following ECS task features:

- User-specified volumes from the EC2 host. You can specify services such as EFS that will mount a volume in your container for you, however.
- Extra networking items such as manually specifying DNS Servers, DNS Search Domains, and extra hosts in the /etc/hosts file
- Task definition options such as specifying an entry point, command, or working directory. These options are available in your Dockerfile and can be specified there.

Important: This service only offers limited tagging support. ECS resources will not be tagged, but any load balancers, EC2 instances, and the Cloudformation stack used to create them will be. See *Tagging Unsupported Resources*.

26.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *ecs* for this service type.
containers	*Containers*	Yes		This section allows you to configure one or more containers that will make up your service.
auto_scaling	*AutoScaling*	Yes		This section contains information about scaling your tasks up and down.
cluster	*Cluster*	No		This section contains items used to configure your ECS cluster of EC2 instances.
load_balancer	*LoadBalancer*	No		If your task needs routing from a load balancer, this section can be used to configure the load balancer's options.
logging	string	No	enabled	Turns CloudWatch logging on or off. Must be either "enabled" or "disabled". See *Logging* for more.
log_retention_in_days	number	No	0	Configures the log retention duration for CloudWatch logs. If set to *0*, logs are kept indefinitely.
tags	*Resource Tags*	No		This section allows you to specify any tags you wish to apply to your ECS service.

26.2.1 Containers

The *containers* section is defined by the following schema:

```
containers:
- name: <string> # Required
  image_name: <string> # Optional
  port_mappings: # Optional, required if you specify 'routing'
  - <integer>
  max_mb: <integer> # Optional. Default: 128
  cpu_units: <integer> # Optional. Default: 100
  links: # Optional
  - <string> # Each value in the list should be the "name" field of another container␣
↪in your containers list
  routing: # Optional
    base_path: <string> # Required
    health_check_path: <string> # Optional. Default: /
  environment_variables: # Optional
    <string>: <string>
```

Note: You may currently only specify the *routing* section in a single container. Attempting to add routing to multiple containers in a single service will result in an error. This is due to a current limitation in the integration between Application Load Balancers (ALB) and ECS that only allows you to attach an ALB to a single container in your task.

Container Image Names

In each container, you may specify an optional *image_name*. If you want to pull a public image from somewhere like DockerHub, just reference the image name:

```
dsw88/my-cool-image
```

If you want to reference an image in your AWS account's EC2 Container Registry (ECR), reference it like this:

```
# The <account> piece will be replaced with your account's long ECR repository name
<account>/my-cool-image
```

If you don't specify an *image_name*, Handel will automatically choose an image name for you based on your Handel naming information. It will use the following image naming pattern:

```
<appName>-<serviceName>-<containerName>:<environmentName>
```

For example, if you don't specify an *image_name* in the below *Example Handel Files*, the two images ECS looks for would be named the following:

```
my-ecs-app-webapp-mywebapp:dev
my-ecs-app-webapp-myothercontainer:dev
```

26.2.2 AutoScaling

The *auto_scaling* section is defined by the following schema:

```
auto_scaling:
  min_tasks: <integer> # Required
  max_tasks: <integer> # Required
  scaling_policies: # Optional
  - type: <up|down> # Required
    adjustment: # Required
      value: <number> # Required
      type: <string> # Optional. Default: 'ChangeInCapacity'. See http://docs.aws.
↪amazon.com/ApplicationAutoScaling/latest/APIReference/API_
↪StepScalingPolicyConfiguration.html for allowed values
      cooldown: <number> # Optional. Default: 300.
    alarm: # Required
      metric_name: <string> # Required
      comparison_operator: <string> # Required. See http://docs.aws.amazon.com/
↪AWSCloudFormation/latest/UserGuide/aws-properties-cw-alarm.html#cfn-cloudwatch-
↪alarms-comparisonoperator for allowed values.
      threshold: <number> # Required
      namespace: <string> # Optional. Default: 'AWS/ECS'
      dimensions: # Optional. Default: Your ECS service dimensions
        <string>: <string>
      period: <number> # Optional. Default: 300
      evaluation_periods: <number> # Optional. Default: 5
```

Tip: Auto-scaling in AWS is based off the CloudWatch service. Configuring auto-scaling can be a bit daunting at first if you haven't used CloudWatch metrics or alarms.

See the below *Example Handel Files* section for some examples of configuring auto-scaling.

Note: If you don't wish to configure auto scaling for your containers, just set *min_tasks* = *max_tasks* and don't configure any *scaling_policies*.

26.2.3 Cluster

The *cluster* section is defined by the following schema:

```
cluster:
  key_name: <string> # Optional. The name of the EC2 keypair to use for SSH access.␣
↪Default: none
  instance_type: <string> # Optional. The type of EC2 instances to use in the cluster.
↪ Default: t2.micro
```

26.2.4 LoadBalancer

The *load_balancer* section is defined by the following schema:

```
load_balancer:
  type: <string> # Required. Allowed values: `http`, `https`.
  timeout: <integer> # Optional. The connection timeout on the load balancer
  https_certificate: <string> # Required if type=https. The ID of the ACM certificate␣
↪to use on the load balancer.
  dns_names:
    - <string> # Optional.
  health_check_grace_period: <integer> # Optional. Default: 15. The period of time,␣
↪in seconds, that the Amazon ECS service scheduler ignores unhealthy Elastic Load␣
↪Balancing target health checks after a task has first started.
```

The *dns_names* section creates one or more dns names that point to this load balancer. See *DNS Records* for more.

26.2.5 Logging

If logging is enabled, a CloudWatch log group will be created, with a name like ecs/<appName>-<environmentName>-<serviceName>. Each container in the container configuration will have a log prefix matching its name. The retention time for the log group is set with *log_retention_in_days*, and defaults to keeping the logs indefinitely.

26.3 Example Handel Files

26.3.1 Simplest Possible ECS Service

This Handel file shows an ECS service with only the required parameters:

```
version: 1

name: my-ecs-app

environments:
  dev:
```

(continues on next page)

(continued from previous page)

```
    webapp:
      type: ecs
      auto_scaling:
        min_tasks: 1
        max_tasks: 1
      containers:
      - name: mywebapp
```

26.3.2 Web Service

This Handel file shows an ECS service configured with HTTP routing to it via a load balancer:

```
version: 1

name: my-ecs-app

environments:
  dev:
    webapp:
      type: ecs
      auto_scaling:
        min_tasks: 1
        max_tasks: 1
      load_balancer:
        type: http
      containers:
      - name: mywebapp
        port_mappings:
        - 5000
        routing:
          base_path: /mypath
          health_check_path: /
```

26.3.3 Multiple Containers

This Handel file shows an ECS service with two containers being configured:

```
version: 1

name: my-ecs-app

environments:
  dev:
    webapp:
      type: ecs
      cluster:
        key_name: mykey
      auto_scaling:
        min_tasks: 1
        max_tasks: 1
      load_balancer:
        type: http
        timeout: 120
```

(continues on next page)

(continued from previous page)

```
      tags:
        mytag: myvalue
      containers:
      - name: mywebapp
        port_mappings:
        - 5000
        max_mb: 256
        cpu_units: 200
        environment_variables:
          MY_VAR: myvalue
        routing:
          base_path: /mypath
          health_check_path: /
      - name: myothercontainer
        max_mb: 256
```

26.3.4 Auto-Scaling On Service CPU Utilization

This Handel file shows an ECS service auto-scaling on its own CPU Utilization metric. Note that in the *alarm* section you can leave off things like *namespace* and *dimensions* and it will default to your ECS service for those values:

```
version: 1

name: my-ecs-app

environments:
  dev:
    webapp:
      type: ecs
      auto_scaling:
        min_tasks: 1
        max_tasks: 11
        scaling_policies:
        - type: up
          adjustment:
            value: 5
          alarm:
            metric_name: CPUUtilization
            comparison_operator: GreaterThanThreshold
            threshold: 70
        - type: down
          adjustment:
            value: 5
          alarm:
            metric_name: CPUUtilization
            comparison_operator: LessThanThreshold
            threshold: 30
      load_balancer:
        type: http
      containers:
      - name: ecstest
        port_mappings:
        - 5000
        routing:
          base_path: /mypath
```

26.3.5 Auto-Scaling On Queue Size

This Handel file shows an ECS service scaling off the size of a queue it consumes:

```
version: 1

name: my-ecs-app

environments:
  dev:
    webapp:
      type: ecs
      auto_scaling:
        min_tasks: 1
        max_tasks: 11
        scaling_policies:
        - type: up
          adjustment:
            value: 5
          alarm:
            namespace: AWS/SQS
            dimensions:
              QueueName: my-ecs-app-dev-queue-sqs
            metric_name: ApproximateNumberOfMessagesVisible
            comparison_operator: GreaterThanThreshold
            threshold: 2000
        - type: down
          adjustment:
            value: 5
          alarm:
            namespace: AWS/SQS
            dimensions:
              QueueName: my-ecs-app-dev-queue-sqs
            metric_name: ApproximateNumberOfMessagesVisible
            comparison_operator: LessThanThreshold
            threshold: 100
      load_balancer:
        type: http
      containers:
      - name: ecstest
        port_mappings:
        - 5000
        routing:
          base_path: /mypath
      dependencies:
      - queue
    queue:
      type: sqs
```

26.4 Depending on this service

The ECS service cannot be referenced as a dependency for another Handel service

26.5 Events produced by this service

The ECS service does not produce events for other Handel services to consume.

26.6 Events consumed by this service

The ECS service does not consume events from other Handel services.

CHAPTER 27

ECS Fargate

This page contains information about the ECS Fargate service supported in Handel. This Handel service provisions your application code as an ECS Fargate Service, with included supporting infrastructure such as load balancers and service auto-scaling groups.

Note: As of February 1, 2017, the AWS Fargate service available in the *us-east-1* region.

27.1 Service Limitations

27.1.1 No EFS Support

Right now you can't consume EFS services as dependencies in this service.

27.1.2 Unsupported ECS task features

This service currently does not support the following ECS task features:

- User-specified volumes from the EC2 host. You can specify services such as EFS that will mount a volume in your container for you, however.
- Extra networking items such as manually specifying DNS Servers, DNS Search Domains, and extra hosts in the /etc/hosts file
- Task definition options such as specifying an entry point, command, or working directory. These options are available in your Dockerfile and can be specified there.

Important: This service only offers limited tagging support. ECS resources will not be tagged, but any load balancers and the Cloudformation stack used to create them will be. See *Tagging Unsupported Resources*.

27.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *ecs-fargate* for this service type.'
max_mb	integer	No	512	The max total MB for all containers in your service. Valid values can be found here
cpu_units	integer	No	256	The max CPU units to use for all containers in your service. Valid values can be found here
containers	*Containers*	Yes		This section allows you to configure one or more containers that will make up your service.
auto_scaling	*AutoScaling*	Yes		This section contains information about scaling your tasks up and down.
load_balancer	*LoadBalancer*	No		If your task needs routing from a load balancer, this section can be used to configure the load balancer's options.
logging	string	No	enabled	Turns CloudWatch logging on or off. Must be either "enabled" or "disabled". See *Logging* for more.
log_retention_in_days	number	No	30	Configures the log retention duration for CloudWatch logs.
tags	*Resource Tags*	No		This section allows you to specify any tags you wish to apply to your ECS service.

27.2.1 Containers

The *containers* section is defined by the following schema:

```
containers:
- name: <string> # Required
  image_name: <string> # Optional
  port_mappings: # Optional, required if you specify 'routing'
  - <integer>
  links: # Optional
  - <string> # Each value in the list should be the "name" field of another container␣
↪in your containers list
  routing: # Optional
    base_path: <string> # Required
    health_check_path: <string> # Optional. Default: /
  environment_variables: # Optional
    <string>: <string>
```

Note: You may currently only specify the *routing* section in a single container. Attempting to add routing to multiple containers in a single service will result in an error. This is due to a current limitation in the integration between Application Load Balancers (ALB) and ECS that only allows you to attach an ALB to a single container in your task.

Container Image Names

In each container, you may specify an optional *image_name*. If you want to pull a public image from somewhere like DockerHub, just reference the image name:

```
dsw88/my-cool-image
```

If you want to reference an image in your AWS account's EC2 Container Registry (ECR), reference it like this:

```
# The <account> piece will be replaced with your account's long ECR repository name
<account>/my-cool-image
```

If you don't specify an *image_name*, Handel will automatically choose an image name for you based on your Handel naming information. It will use the following image naming pattern:

```
<appName>-<serviceName>-<containerName>:<environmentName>
```

For example, if you don't specify an *image_name* in the below *Example Handel Files*, the two images ECS looks for would be named the following:

```
my-ecs-app-webapp-mywebapp:dev
my-ecs-app-webapp-myothercontainer:dev
```

27.2.2 AutoScaling

The *auto_scaling* section is defined by the following schema:

```
auto_scaling:
  min_tasks: <integer> # Required
  max_tasks: <integer> # Required
  scaling_policies: # Optional
  - type: <up|down>
    adjustment:
      type: <string> # Optional. Default: 'ChangeInCapacity'. See http://docs.aws.
↪amazon.com/ApplicationAutoScaling/latest/APIReference/API_
↪StepScalingPolicyConfiguration.html for allowed values
      value: <number> # Required
      cooldown: <number> # Optional. Default: 300.
    alarm:
      namespace: <string> # Optional. Default: 'AWS/ECS'
      dimensions: # Optional. Default: Your ECS service dimensions
        <string>: <string>
      metric_name: <string> # Required
      comparison_operator: <string> # Required. See http://docs.aws.amazon.com/
↪AWSCloudFormation/latest/UserGuide/aws-properties-cw-alarm.html#cfn-cloudwatch-
↪alarms-comparisonoperator for allowed values.
      threshold: <number> # Required
      period: <number> # Optional. Default: 300
      evaluation_periods: <number> # Optional. Default: 5
```

Tip: Auto-scaling in AWS is based off the CloudWatch service. Configuring auto-scaling can be a bit daunting at first if you haven't used CloudWatch metrics or alarms.

See the below *Example Handel Files* section for some examples of configuring auto-scaling.

Note: If you don't wish to configure auto scaling for your containers, just set *min_tasks* = *max_tasks* and don't configure any *scaling_policies*.

27.2.3 LoadBalancer

The *load_balancer* section is defined by the following schema:

```
load_balancer:
  type: <string> # Required. Allowed values: `http`, `https`.
  timeout: <integer> # Optional. The connection timeout on the load balancer
  https_certificate: <string> # Required if type=https. The ID of the ACM certificate␣
↪to use on the load balancer.
  dns_names:
   - <string> # Optional.
  health_check_grace_period: <integer> # Optional. Default: 15. The period of time,␣
↪in seconds, that the Amazon ECS service scheduler ignores unhealthy Elastic Load␣
↪Balancing target health checks after a task has first started.
```

The *dns_names* section creates one or more dns names that point to this load balancer. See *DNS Records* for more.

27.2.4 Logging

If logging is enabled, a CloudWatch log group will be created, with a name like fargate/<appName>-<environmentName>-<serviceName>. Each container in the container configuration will have a log prefix matching its name. The retention time for the log group is set with *log_retention_in_days*, and defaults to keeping the logs indefinitely.

27.3 Example Handel Files

27.3.1 Simplest Possible Fargate Service

This Handel file shows an ECS service with only the required parameters:

```
version: 1

name: my-fargate-app

environments:
  dev:
    webapp:
      type: ecs-fargate
      auto_scaling:
        min_tasks: 1
        max_tasks: 1
      containers:
      - name: mywebapp
```

27.3.2 Web Service

This Handel file shows a Fargate service configured with HTTP routing to it via a load balancer:

```
version: 1

name: my-fargate-app

```

(continues on next page)

(continued from previous page)

```
environments:
  dev:
    webapp:
      type: ecs-fargate
      auto_scaling:
        min_tasks: 1
        max_tasks: 1
      load_balancer:
        type: http
      containers:
      - name: mywebapp
        port_mappings:
        - 5000
        routing:
          base_path: /mypath
          health_check_path: /
```

27.3.3 Multiple Containers

This Handel file shows a Fargate service with two containers being configured:

```
version: 1

name: my-fargate-app

environments:
  dev:
    webapp:
      type: ecs-fargate
      auto_scaling:
        min_tasks: 1
        max_tasks: 1
      load_balancer:
        type: http
        timeout: 120
      tags:
        mytag: myvalue
      containers:
      - name: mywebapp
        port_mappings:
        - 5000
        environment_variables:
          MY_VAR: myvalue
        routing:
          base_path: /mypath
          health_check_path: /
      - name: myothercontainer
```

27.3.4 Auto-Scaling On Service CPU Utilization

This Handel file shows a Fargate service auto-scaling on its own CPU Utilization metric. Note that in the *alarm* section you can leave off things like *namespace* and *dimensions* and it will default to your Fargate service for those values:

```
version: 1

name: my-fargate-app

environments:
  dev:
    webapp:
      type: ecs-fargate
      auto_scaling:
        min_tasks: 1
        max_tasks: 11
        scaling_policies:
        - type: up
          adjustment:
            value: 5
          alarm:
            metric_name: CPUUtilization
            comparison_operator: GreaterThanThreshold
            threshold: 70
        - type: down
          adjustment:
            value: 5
          alarm:
            metric_name: CPUUtilization
            comparison_operator: LessThanThreshold
            threshold: 30
      load_balancer:
        type: http
      containers:
      - name: fargatetest
        port_mappings:
        - 5000
        routing:
          base_path: /mypath
```

27.3.5 Auto-Scaling On Queue Size

This Handel file shows an ECS service scaling off the size of a queue it consumes:

```
version: 1

name: my-fargate-app

environments:
  dev:
    webapp:
      type: ecs-fargate
      auto_scaling:
        min_tasks: 1
        max_tasks: 11
        scaling_policies:
        - type: up
          adjustment:
            value: 5
          alarm:
            namespace: AWS/SQS
```

(continues on next page)

(continued from previous page)

```
          dimensions:
            QueueName: my-fargate-app-dev-queue-sqs
          metric_name: ApproximateNumberOfMessagesVisible
          comparison_operator: GreaterThanThreshold
          threshold: 2000
      - type: down
        adjustment:
          value: 5
        alarm:
          namespace: AWS/SQS
          dimensions:
            QueueName: my-fargate-app-dev-queue-sqs
          metric_name: ApproximateNumberOfMessagesVisible
          comparison_operator: LessThanThreshold
          threshold: 100
    load_balancer:
      type: http
    containers:
    - name: fargatetest
      port_mappings:
      - 5000
      routing:
        base_path: /mypath
    dependencies:
    - queue
  queue:
    type: sqs
```

27.4 Depending on this service

The ECS Fargate service cannot be referenced as a dependency for another Handel service

27.5 Events produced by this service

The ECS Fargate service does not produce events for other Handel services to consume.

27.6 Events consumed by this service

The ECS Fargate service does not consume events from other Handel services.

CHAPTER 28

EFS (Elastic File System)

This page contains information about using the EFS (Elastic File System) service in Handel. This service provides an EFS mount for use by other compute services such as ElasticBeanstalk and ECS.

28.1 Service Limitations

28.1.1 No Update Support

This service intentionally does not support updates. Once a file system is created, updates to it (like changing the performance mode) will cause a new file system to be created and the old one deleted. In an effort to avoid unwanted data loss, we don't update this service automatically.

28.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *efs* for this service type.
performance_mode	string	No	general_purpose	What kind of performance for the EFS mount. Allowed values: general_purpose, max_io
tags	*Resource Tags*	No		Any tags you wish to apply to this EFS mount.

28.3 Example Handel File

```
version: 1

name: my-efs-app

environments:
  dev:
    webapp:
      type: efs
      performance_mode: general_purpose
      tags:
        mytag: myvalue
```

28.4 Depending on this service

The EFS service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_MOUNT_DIR	The directory on the host where the EFS volume was mounted.

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

28.5 Events produced by this service

The EFS service does not produce events for other Handel services to consume.

28.6 Events consumed by this service

The EFS service does not consume events from other Handel services.

CHAPTER 29

Elasticsearch

This page contains information about using the Elasticsearch service in Handel. This service provides an Amazon ElasticSearch cluster.

> **Warning:** This provisioner is new and should be considered in beta. It is subject to breaking changes until this beta label is removed.

29.1 Service Limitations

29.1.1 No Zone Awareness Support

Currently Elasticsearch clusters are only deployed in a single Availability Zone (AZ), and there is no support for the two-AZ zone awareness support.

29.1.2 No Kibana Support

While Kibana is deployed with the Elasticsearch cluster, there is currently no way for you to access it since the cluster does not have wide-open security permissions and Cognito authentication isn't supported.

29.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *elasticsearch* for this service type.
version	number	Yes		The version number of ElasticSearch to use. See Supported Elasticsearch Versions for more details
instance_type	string	No	t2.small.elasticsearch	The size of database instance to run. See Elasticsearch Pricing for the allowed instance types.
instance_count	number	No	1	The number of instances to run in your cluster.
ebs	*EBS*	No		This section is required if you specify an instance type that uses EBS storage instead of the instance store.
master_node	*MasterNode*	No		If you specify this section, you will configure a master node cluster to handle cluster management operations.
tags	*Resource Tags*	No		Any tags you wish to apply to this Elasticsearch cluster.

29.2.1 EBS

The *ebs* section is defined by the following schema:

```
ebs:
  size_gb: <number> # Required. The size of the EBS disk in GB
  provisioned_iops: <number> # Optional. The number of provisioned IOPS you want to␣
↪dedicate to the EBS disk.
```

Important: Each instance type has different values for the allowed values of the *size_gb* parameter. See EBS Volume Size Limits for the allowed values for each instance type

29.2.2 MasterNode

The *master_node* section is defined by the following schema:

```
master_node:
  instance_type: <string> # Required
  instance_count: <number> # Required
```

Note: Amazon recommends using master nodes to increase cluster stability. See Dedicated Master Nodes for their recommendations.

29.3 IAM Authentication

Your ElasticSearch cluster requires IAM authentication to your Elasticsearch endpoint. This is done using AWS' signature version 4 signing process. Each HTTP request to Elasticsearch must include the signature headers required

by AWS to validate your IAM role identity.

See AWS' Programmatic Indexing page for information about how perform this authentication in various languages.

29.4 Example Handel File

```
version: 1

name: elasticsearch-test

environments:
  dev:
    search:
      type: elasticsearch
      version: 6.2
      instance_type: t2.small.elasticsearch
      instance_count: 1
      ebs:
        size_gb: 10
```

29.5 Depending on this service

The Elasticsearch service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_DOMAIN_ENDPOINT	The address that you should use to communicate with the cluster.
<SERVICE_NAME>_DOMAIN_NAME	The name of your Elasticsearch domain.

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

29.6 Events produced by this service

The Elasticsearch service does not produce events for other Handel services to consume.

29.7 Events consumed by this service

The Elasticsearch service does not consume events from other Handel services.

CHAPTER 30

IoT

This document contains information about the IoT service supported in Handel. This Handel service currently provisions IoT topic rules that can invoke things like Lambda functions.

30.1 Service Limitations

This Handel service is quite new, and as such doesn't support all of IoT yet. In particular, the following are not supported:

- Creating IoT Things.
- Creating IoT Certificates.
- Creating IoT Policies.

Important: This service only offers limited tagging support. IoT resources will not be tagged, but the Cloudformation stack used to create them will be. See *Tagging Unsupported Resources*.

30.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *iot* for this service type.
description	string	No	AWS IoT rule created by Handel	The description you would like to be applied to the IoT rule.
tags	*Resource Tags*	No		Tags to be applied to the Cloudformation stack which provisions this resource.

30.3 Example Handel File

The following example shows setting up an IoT topic rule to produce to a Lambda:

```
version: 1

name: my-topic-rule

environments:
  dev:
    topicrule:
      type: iot
      event_consumers:
      - service_name: function
        sql: "select * from 'something';"
    function:
      type: lambda
      path_to_code: .
      handler: index.handler
      runtime: nodejs6.10
```

30.4 Depending on this service

The IoT service cannot currently be specified as a dependency by any other services. It is currently only functioning as an event producer for other services such as Lambda.

30.5 Events produced by this service

The IoT service can produce events to the following service types:

- Lambda

30.5.1 Event consumer parameters

When specifying event consumers on the IoT service, you may specify the following parameters:

Parameter	Type	Required	Default	Description
service_name	string	Yes		This is the name of the service in your Handel file to which you would like to produce events.
sql	string	Yes		This is where you specify the IoT-compatible SQL statement that will cause your rule to fire.
description	string	No	AWS IoT rule created by Handel.	The description for the topic rule payload.
rule_disabled	boolean	No	false	This defines whether the topic rule is currently enabled or disabled.

30.6 Events consumed by this service

The IoT service cannot currently consume events from other services.

CHAPTER 31

KMS (Key Management Service)

This document contains information about the KMS service supported in Handel. This Handel service provisions a KMS key and alias for use by your applications.

31.1 Service Limitations

This service currently does not allow creating disabled keys. It also uses IAM instead of custom Key Policies to control access to the key, as key policies can easily make keys unmanageable.

While the AWS API allows for multiple aliases to point to a single key, this service matches the AWS Console in enforcing a one-to-one relationship between keys.

Important: This service only offers limited tagging support. KMS Keys will not be tagged, but the Cloudformation stack used to create them will be. See *Tagging Unsupported Resources*.

31.2 Parameters

This service takes the following parameters:

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *kms* for this service type.
alias	string	No	<appName>/<environmentName>/<serviceName>	The name of the alias to create. This name must be unique across the account and region in which the key is deployed.
auto_rotate	boolean	No	true	Whether to allow AWS to auto-rotate the underlying Master Key.
tags	*Resource Tags*	No		Tags to be applied to the Cloudformation stack which provisions this resource.

31.3 Example Handel File

This Handel file shows a KMS key being configured:

```
version: 1

name: my-app

environments:
  dev:
    mykey:
      type: kms
      # because we don't specify an alias, the alias will be my-app/dev/mykey (see
↪above)
      auto_rotate: true
```

31.4 Depending on this service

This service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_KEY_ID	The id of the created key
<SERVICE_NAME>_KEY_ARN	The ARN of the created key
<SERVICE_NAME>_ALIAS_NAME	The name of the created alias
<SERVICE_NAME>_ALIAS_ARN	The ARN of the created alias

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

31.5 Events produced by this service

The KMS service does not currently produce events for other Handel services. Support for producing events upon key rotation is planned for the future.

31.6 Events consumed by this service

The KMS service does not consume events from other Handel services.

CHAPTER 32

Lambda

This document contains information about the Lambda service supported in Handel. This Handel service provisions an Lambda function. You can reference this function in other services as an event consumer, which will invoke the function when events occur.

32.1 Service Limitations

The following Lambda features are not currently supported in this service:

- Encrypting environment variables with KMS keys

32.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *lambda* for this service type.
path_to_code	string	Yes		The location of your code to upload to Lambda. This can be a directory (which will be zipped up) or a single file (such as a deployable Java WAR file or pre-existing zip file)
handler	string	Yes		The handler function in your code that is the entry-point to the Lambda.
runtime	string	Yes		The Lambda runtime that will execute your code
description	string	No	Handel-created function	The configuration description of your function
memory	number	No	128	The amount of memory to allocate for your function
timeout	number	No	3	The timeout in seconds for your function. Max 300
vpc	boolean	No	false	If true, the lambda will be deployed inside your VPC. Inside your VPC, it will be able to communicate with resources like RDS databases and ElastiCache clusters.
environment_variables	*EnvironmentVariables*	No		Any environment variables you want to inject into your code.
tags	*Resource Tags*	No		Any tags you want to apply to your Lambda

32.2.1 EnvironmentVariables

The EnvironmentVariables element is defined by the following schema:

```
environment_variables:
  <YOUR_ENV_NAME>: <your_env_value>
```

<YOUR_ENV_NAME> is a string that will be the name of the injected environment variable. <your_env_value> is its value. You may specify an arbitrary number of environment variables in this section.

32.3 Example Handel File

```
version: 1

name: my-lambda

environments:
  dev:
    webapp:
      type: lambda
      path_to_code: .
      handler: index.handler
      runtime: nodejs6.10
```

(continues on next page)

(continued from previous page)

```
    environment_variables:
      MY_ENV: myEnvValue
    tags:
      mytag: mytagvalue
```

32.4 Running a scheduled Lambda

To run a scheduled Lambda, you can use this service in conjunction with the CloudWatch Events service. See the *Scheduled Lambda* on the CloudWatch Events service for details on how to do this.

32.5 Depending on this service

The Lambda service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_FUNCTION_NAME	The name of the created Lambda function
<SERVICE_NAME>_FUCNTION_ARN	The ARN of the created Lambda function

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

32.6 Events produced by this service

The Lambda service does not currently produce events for other Handel services to consume.

32.7 Events consumed by this service

The Lambda service can consume events from the following service types:

- Alexa Skill Kit
- CloudWatch Events
- DynamoDB
- IoT
- S3
- SNS

CHAPTER 33

Memcached (ElastiCache)

This page contains information about using the Memcached service in Handel. This service provides a Memcached cluster via the ElastiCache service.

33.1 Service Limitations

33.1.1 No Scheduled Maintenance Window Configuration

This service currently doesn't allow you to change the maintenance window for your Memcached cluster.

33.1.2 No Snapshot Window Configuration

This service currently doesn't allow you to change the snapshot window for your Memcached cluster.

33.1.3 No Restoration From Snapshot

This service currently doesn't allow you to launch a cluster from a previous cluster snapshot.

33.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *memcached* for this service type.
instance_type	string	Yes		The size of each Memcached instance in your cluster. See Choosing Your Node Size for more details.
memcached_version	string	Yes		The version of Memcached to run. See Comparing Memcached Versions for a list of available versions.
description	string	No	Parameter group for cluster.	The parameter group description of your cluster.
node_count	number	No	1	The number of memcached nodes you want in your cluster.
cache_parameters	Map<string,string>	No		Any cache parameters you wish for your Memcached cluster. See Memcached Specific Parameters for the list of parameters you can provide.
tags	*Resource Tags*	No		Any tags you wish to apply to this Memcached cluster.

Warning: Note that having more than 1 node in your cluster will greatly increase your cost. Each node you add to the cluster adds a full cache instance type node cost to your cluster cost.

For example, if you have a Memcached cluster of size 1, using a cache.m4.large instance, it will cost about $112/month.

If you have that same cache.m4.large type, but with a cluster size of 4, it will cost about $448/month since you are being charged for four full Memcached instances.

Be careful to calculate how much this service will cost you if you are using a cluster of more than 1 node.

33.3 Example Handel File

```
version: 1

name: my-memcached-cluster

environments:
  dev:
    cache:
      type: memcached
      instance_type: cache.m3.medium
      memcached_version: 1.4.34
      node_count: 1
      cache_parameters:
        cas_disabled: 1
      tags:
        mytag: myvalue
```

33.4 Depending on this service

The Memcached service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_ADDRESS	The DNS name of the Memcached configuration endpoint address.
<SERVICE_NAME>_PORT	The port on which the Memcached cluster is listening.

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

33.5 Events produced by this service

The Memcached service does not produce events for other Handel services to consume.

33.6 Events consumed by this service

The Memcached service does not consume events from other Handel services.

CHAPTER 34

MySQL (RDS)

This page contains information about using the MySQL service in Handel. This service provides a MySQL database via the RDS service.

34.1 Service Limitations

34.1.1 No Option Group Support

This service doesn't allow you to specify any custom options in an option group. It does allow you specify custom parameters in a parameter group, however.

34.1.2 No Update Support

This service intentionally does not support updates. Once a database is created, certain updates to the database will cause a new database to be created and the old one deleted. In an effort to avoid unwanted data loss, we don't update this service automatically. You can still modify the database and parameter group manually in the AWS console.

Warning: Make sure you know what you're doing when you modify your RDS database in the AWS Console. Certain actions will cause database downtime, and some may even cause the database to be recreated.

34.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *mysql* for this service type.
mysql_version	string	Yes		The version of MySQL you wish to run. See MySQL on Amazon RDS for the list of supported versions.
database_name	string	Yes		The name of your database in your MySQL instance.
description	string	No	Parameter group.	The parameter group description.
instance_type	string	No	db.t2.micro	The size of database instance to run. See DB Instance Class for information on choosing an instance type.
storage_gb	number	No	5	The number of Gigabytes (GB) of storage to allocate to your database.
storage_type	string	No	standard	The type of storage to use, whether magnetic or SSD. Allowed values: 'standard', 'gp2'.
multi_az	boolean	No	false	Whether or not the deployed database should be Multi-AZ. Note: Using Multi-AZ increases the cost of your database.
db_parameters	map<string, string>	No		A list of key/value MySQL parameter group pairs to configure your database. You will need to look in the AWS Console to see the list of available parameters for MySQL.
tags	*Resource Tags*	No		Any tags you wish to apply to this MySQL instance.

Warning: Be aware that large database instances are very expensive. The *db.cr1.8xl* instance type, for example, costs about $3,400/month. Make sure you check how much you will be paying!

You can use the excellent EC2Instances.info site to easily see pricing information for RDS databases.

34.3 Example Handel File

```
version: 1

name: my-mysql-instance

environments:
  dev:
    database:
      type: mysql
      database_name: mydb
      instance_type: db.t2.micro
      storage_gb: 5
      mysql_version: 5.6.27
      storage_type: standard
      db_parameters:
        autocommit: 1
```

(continues on next page)

(continued from previous page)

```
    tags:
      mytag: myvalue
```

34.4 Depending on this service

The MySQL service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_ADDRESS	The DNS name of the MySQL database address.
<SERVICE_NAME>_PORT	The port on which the MySQL instance is listening.
<SERVICE_NAME>_DATABASE_NAME	The name of the database in your MySQL instance.

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

In addition, the MySQL service puts the following credentials into the EC2 parameter store:

Parameter Name	Description
<parameter_prefix>.<service_name>.db_username	The username for your database user.
<parameter_prefix>.<service_name>.db_password	The password for your database user.

Note: The <parameter_prefix> section of the parameter name is a consistent prefix applied to all parameters injected by services in the EC2 Parameter Store. See *Parameter Store Prefix* for information about the structure of this prefix.

The <service_name> section of the parameter name should be replaced by the *service name* you gave your database in your Handel file.

34.5 Events produced by this service

The MySQL service does not produce events for other Handel services to consume.

34.6 Events consumed by this service

The MySQL service does not consume events from other Handel services.

CHAPTER 35

Neptune

This page contains information about using the Neptune service in Handel. This service provides a Neptune graph database cluster.

Warning: This provisioner is new and should be considered in beta. It is subject to breaking changes until this beta label is removed.

35.1 Service Limitations

35.1.1 No Update Support

This service intentionally does not support updates. Once a database is created, certain updates to the database will cause a new database to be created and the old one deleted. In an effort to avoid unwanted data loss, we don't update this service automatically. You can still modify the database and parameter group manually in the AWS console.

Warning: Make sure you know what you're doing when you modify your Neptune database in the AWS Console. Certain actions will cause database downtime, and some may even cause the database to be recreated.

35.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *neptune* for this service type.
description	string	No		The description on the resources created for the cluster
instance_type	string	No	db.r4.large	The size of database instance to run. See Neptune pricing for the allowed instance types.
cluster_size	number	No	1	The number of instances (including the primary) to run in your cluster.
iam_auth_enabled	boolean	No	true	Whether your Neptune cluster should have IAM authentication enabled. NOTE: If you specify false, your Neptune instance will be writeable by anyone inside your VPC.
cluster_parameters	map<string, string>	No		A list of key/value Neptune cluster parameter group pairs to configure your cluster. You will need to look in the AWS Console to see the list of available cluster parameters for Neptune.
instance_parameters	map<string, string>	No		A list of key/value Neptune instance parameter group pairs to configure the instances in your cluster. You will need to look in the AWS Console to see the list of available instance parameters for Neptune.
tags	*Resource Tags*	No		Any tags you wish to apply to this Neptune cluster.

Warning: Be aware that Neptune clusters can be very expensive. A cluster with 3 *db.r4.2xlarge* instances in it will cost about about $3,000/month. Make sure you check how much you will be paying!

35.3 Example Handel File

```
version: 1

name: neptune-test

environments:
  dev:
    database:
      type: neptune
      instance_type: db.r4.large
      cluster_size: 3
      cluster_parameters: # This is where you can set parameters that configure the␣
↪cluster as a whole
        neptune_enable_audit_log: 0
      instance_parameters: # This is where you can set parameters that apply to each␣
↪instance.
        neptune_query_timeout: 120000
      tags:
        some: tag
```

35.4 Depending on this service

The Neptune service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_CLUSTER_ENDPOINT	The address that you should use for writes to the database.
<SERVICE_NAME>_READ_ENDPOINT	The address that you should use for reads to the database.
<SERVICE_NAME>_PORT	The port on which the Neptune cluster instances are listening.

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

35.5 Events produced by this service

The Neptune service does not produce events for other Handel services to consume.

35.6 Events consumed by this service

The Neptune service does not consume events from other Handel services.

CHAPTER 36

PostgreSQL (RDS)

This page contains information about using the PostgreSQL service in Handel. This service provides a PostgreSQL database via the RDS service.

36.1 Service Limitations

36.1.1 No Option Group Support

This service doesn't allow you to specify any custom options in an option group. It does allow you specify custom parameters in a parameter group, however.

36.1.2 No Update Support

This service intentionally does not support updates. Once a database is created, certain updates to the database will cause a new database to be created and the old one deleted. In an effort to avoid unwanted data loss, we don't update this service automatically. You can still modify the database and parameter group manually in the AWS console.

Warning: Make sure you know what you're doing when you modify your RDS database in the AWS Console. Certain actions will cause database downtime, and some may even cause the database to be recreated.

36.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *postgresql* for this service type.
database_name	string	Yes		The name of your database in your PostgreSQL instance.
postgres_version	string	Yes		The version of PostgreSQL you wish to run. See PostgreSQL on Amazon RDS for the list of supported versions.
description	string	No	Parameter group.	The parameter group description.
instance_type	string	No	db.t2.micro	The size of database instance to run. See DB Instance Class for information on choosing an instance type.
storage_gb	number	No	5	The number of Gigabytes (GB) of storage to allocate to your database.
storage_type	string	No	standard	The type of storage to use, whether magnetic or SSD. Allowed values: 'standard', 'gp2'.
multi_az	boolean	No	false	Whether or not the deployed database should be Multi-AZ. Note: Using Multi-AZ increases the cost of your database.
db_parameters	map<string, string>	No		A list of key/value PostgreSQL parameter group pairs to configure your database. You will need to look in the AWS Console to see the list of available parameters for PostgreSQL.
tags	*Resource Tags*	No		Any tags you wish to apply to this PostgreSQL instance.

Warning: Be aware that large database instances are very expensive. The *db.cr1.8xl* instance type, for example, costs about $3,400/month. Make sure you check how much you will be paying!

You can use the excellent EC2Instances.info site to easily see pricing information for RDS databases.

36.3 Example Handel File

```
version: 1

name: my-postgres-instance

environments:
  dev:
    database:
      type: postgresql
      database_name: mydb
      instance_type: db.t2.micro
      storage_gb: 5
      postgres_version: 9.6.2
      storage_type: standard
      db_parameters:
        authentication_timeout: 600
```

(continues on next page)

(continued from previous page)

```
    tags:
      mytag: myvalue
```

36.4 Depending on this service

The PostgreSQL service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_ADDRESS	The DNS name of the PostgreSQL database address.
<SERVICE_NAME>_PORT	The port on which the PostgreSQL instance is listening.
<SERVICE_NAME>_DATABASE_NAME	The name of the database in your PostgreSQL instance.

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

In addition, the PostgreSQL service puts the following credentials into the EC2 parameter store:

Parameter Name	Description
<parameter_prefix>.<service_name>.db_username	The username for your database user.
<parameter_prefix>.<service_name>.db_password	The password for your database user.

Note: The <parameter_prefix> section of the parameter name is a consistent prefix applied to all parameters injected by services in the EC2 Parameter Store. See *Parameter Store Prefix* for information about the structure of this prefix.

The <service_name> section of the parameter name should be replaced by the *service name* you gave your database in your Handel file.

36.5 Events produced by this service

The PostgreSQL service does not produce events for other Handel services to consume.

36.6 Events consumed by this service

The PostgreSQL service does not consume events from other Handel services.

CHAPTER 37

Redis (ElastiCache)

This page contains information about using the Redis service in Handel. This service provides a Redis cluster via the ElastiCache service.

37.1 Service Limitations

37.1.1 No Cluster Mode Support

This service currently does not support using Redis in cluster mode. It does support replication groups with a primary node and 1 or more read replicas, but it doesn't yet support Redis' cluster mode sharding.

37.1.2 No Restoration From Snapshot

This service currently doesn't allow you to launch a cluster from a previous cluster snapshot.

37.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *redis* for this service type.
instance_type	string	Yes		The size of each Redis instance in your cluster. See Choosing Your Node Size for more details.
redis_version	string	Yes		The version of Redis to run. See Comparing Redis Versions for a list of available versions.
description	string	No	Parameter group.	The redis group description.
maintenance_window	string	No		The weekly time range (in UTC) during which ElastiCache may peform maintenance on the node group. For example, you can specify Sun:05:00-Tue:09:00.
read_replicas	number	No	0	The number of read replicas you want to provision. Allowed values: 0-5.
snapshot_window	string	No		The daily time range (in UTC) during which ElastiCache will begin taking a daily snapshot of your node group. For example, you can specify 05:00-09:00. This feature is not available on the t2 and t1 instance types.
cache_parameters	Map<string,string>	No		Any cache parameters you wish for your Redis cluster. See Redis Specific Parameters for the list of parameters you can provide.
tags	*Resource Tags*	No		Any tags you wish to apply to this Redis cluster.

Warning: If you use read replicas, be aware that it will greatly increase your cost. Each read replica you use adds the full cost of another Redis node.

For example, if you have a single cache.m4.large Redis instance with no read replicas, it will cost about $112/month.

If you have that same cache.m4.large type, but with 1 read replica, it will cost you double at about $224/month since you are being charged for two full Redis instances.

Taken to its extreme, a cache.m4.large with 5 read replicas will cost about $673/month. **Be careful to calculate how much this service will cost you if you are using read replicas**

37.3 Example Handel File

```
version: 1

name: my-redis-cluster

environments:
  dev:
    cache:
      type: redis
```

(continues on next page)

(continued from previous page)

```
    instance_type: cache.m3.medium
    redis_version: 3.2.4
    read_replicas: 1
    cache_parameters:
      activerehashing: 'no'
    tags:
      mytag: myvalue
```

37.4 Depending on this service

The Redis service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_ADDRESS	The DNS name of the primary Redis node
<SERVICE_NAME>_PORT	The port on which the primary Redis node is listening.

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

37.5 Events produced by this service

The Redis service does not produce events for other Handel services to consume.

37.6 Events consumed by this service

The Redis service does not consume events from other Handel services.

CHAPTER 38

Route 53 Hosted Zone

This document contains information about the Route 53 Hosted Zone service supported in Handel. This Handel service provisions a Route 53 Hosted Zone, in which you can create other DNS records.

38.1 Service Limitations

The following Route 53 features are not currently supported in this service:

- Domain Name Registration

38.2 Manual Steps

If you are creating a public zone as a subdomain of another domain (like myapp.mydomain.com), you must register it with your DNS provider.

If you are using Handel for your work at a company or organization of some kind, they likely have a process for registering these hosted zones with their DNS provider. Check with the networking groups in your organization to find out how you can do this.

38.3 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *route53zone* for this service type.
name	string	Yes		The DNS name for this hosted zone.
private	boolean	No	false	Whether or not this is a private zone. If it is a private zone, it is only accessible by the VPC in your account config file.
tags	*Resource Tags*	No		Any tags you want to apply to your Hosted Zone

38.4 Example Handel File

```
version: 1

name: my-dns

environments:
  dev:
    public-zone:
      type: route53zone
      name: mydomain.example.com
      tags:
        mytag: mytagvalue
    private-zone:
      type: route53zone
      name: private.myapp # Doesn't have to have a normal top-level domain
      private: true
      tags:
        mytag: mytagvalue
```

38.5 Depending on this service

This service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_ZONE_NAME	The DNS name of hosted zone.
<SERVICE_NAME>_ZONE_ID	The id of the hosted zone
<SERVICE_NAME>_ZONE_NAME_SERVERS	A comma-delimited list of the name servers for this hosted zone. For example: ns1.example.com,ns2.example.co.uk

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

38.5.1 DNS Records

Certain supported services can create an alias record in this zone. The currently supported services are:

- *API Gateway*
- *Beanstalk*
- *ECS*
- *ECS (Fargate)*
- *S3 Static Site*

API Gateway, Beanstalk, ECS, and ECS (Fargate) can support multiple DNS entries.

See the individual service documentation for how to define the DNS names.

The DNS name must either match or be a subdomain of an existing Route 53 hosted zone name. If the hosted zone is configured in the same Handel environment, you must declare it as a dependency of the service consuming it, so that Handel can make sure that your resources are constructed in the right order.

```
version: 1

name: my-app

environments:
  dev:
    dns:
      type: route53zone
      name: myapp.example.com
    private-dns:
      type: route53zone
      name: internal.myapp
      private: true
    beanstalk-app:
      type: beanstalk
      routing:
        type: http
        dns_names:
          - beanstalk.mymapp.example.com
      ...
      dependencies:
        - dns
    ecs-app:
      type: ecs
      load_balancer:
        type: http
        dns_names:
          - ecs.myapp.example.com
          - ecs.internal.myapp
      ...
      dependencies:
        - dns
        - private-dns
    another-beanstalk:
      type: beanstalk
      routing:
        type: http
        dns_names:
```

(continues on next page)

(continued from previous page)

```
          - mysite.example.com # This requires that a hosted zone for mysite.example.
↪com have already been configured.
      ...
```

38.6 Events produced by this service

The Route 53 Hosted Zone service does not currently produce events for other Handel services to consume.

38.7 Events consumed by this service

The Route 53 Hosted Zone service does not currently consume events from other Handle services.

CHAPTER 39

S3 (Simple Storage Service)

This document contains information about the S3 service supported in Handel. This Handel service provisions an S3 bucket for use by your applications.

Note: For static websites in S3, see the *S3 Static Site* service.

39.1 Service Limitations

This service currently only provisions a bare-bones S3 bucket for data storage. It does support versioning, but the following other features are not currently supported:

- CORS configuration
- Bucket logging
- Cross-region replication

39.2 Parameters

This service takes the following parameters:

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *s3* for this service type.
bucketName	string	No	<appName>-<environmentName>-<serviceName>-<serviceType>	The name of the bucket to create. This name must be globally unique across all AWS accounts, so 'myBucket' will likely be taken. :)
bucketAcl	string	No		**Warning:** A canned access control list (ACL) that grants predefined permissions to the bucket. These are global permissions ie, PublicRead means the bucket is open to the world. Allowed values: *AuthenticatedRead*, *AwsExecRead*, *BucketOwnerRead*, *BucketOwnerFullControl*, *LogDeliveryWrite*, *Private*, *PublicRead*
versioning	string	No	disabled	Whether to enable versioning on the bucket. Allowed values: *enabled*, *disabled*
logging	string	No	disabled	Whether to enable logging on the bucket. Allowed values: *enabled*, *disabled*.
lifecycles	*Lifecycles*	No		Lifecycle Policies to apply to the bucket. See AWS Docs for more info
tags	*Resource Tags*	No		Any tags you want to apply to your S3 bucket

39.2.1 Lifecycles

A list of life cycle rules

```
lifecycles:
  - name: <string> # Required
    prefix: <string> # Optional
    transitions: # Optional but one of transitions or version_transitions are required
      - type: <ia, glacier, expiration> # type must be ia (Standard-IA infrequent␣
↪Access), glacier, or expiration)
        days: 30
    version_transitions: # Optional but one of transitions or version_transitions are␣
↪required, only days are supported
      - type: <ia, glacier, expiration>
        days: 30
```

Transitions are defined by the following:

Parameter	Type	Required	Default	Description
type	string	Yes	None	Type of transition must be one of ia(Standard Infrequent Access), glacier, expiration (deletion)
days	integer	No	None	Number of days until transition *must specify all transition as days or dates not both*
date	ISO 8601 UTC	No	None	Date to transition in ISO 8602 UTC format *must specify all transition as days or dates not both*

More complex example:

```
lifecycles:
  - name: ia30glacier365expire720
    transitions:
      - type: ia
        days: 30
      - type: expiration
        days: 720
      - type: glacier
        days: 365
    version_transitions:
      - type: ia
        days: 30
      - type: expiration
        days: 90
```

39.3 Example Handel File

39.3.1 Simple Bucket

This Handel file shows an S3 service being configured:

```
version: 1

name: my-s3-bucket

environments:
  dev:
    mybucket:
      type: s3
      # Because we don't specify a bucket_name, the bucket will be named 'my-s3-
↪bucket-dev-mybucket-s3' (see default in table above)
      versioning: enabled
```

39.3.2 S3 Events

This Handel file shows an S3 service that is configured to send events to a Lambda function:

```
version: 1

name: test-s3-events

environments:
  dev:
    function:
      type: lambda
      path_to_code: .
      handler: index.handler
      runtime: python3.6
    bucket:
      type: s3
      event_consumers:
```

(continues on next page)

(continued from previous page)

```
    - service_name: function
      bucket_events:
      - s3:ObjectCreated:*
      filters:
      - name: prefix
        value: somefolderprefix
```

Filters for Bucket Suffixes are also supported.

39.4 Depending on this service

This service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_BUCKET_NAME	The name of the created bucket
<SERVICE_NAME>_BUCKET_URL	The HTTPS URL of the created bucket
<SERVICE_NAME>_REGION_ENDPOINT	The domain of the S3 region endpoint, which you can use when configuring your AWS SDK

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

39.5 Events produced by this service

The CloudWatch Events service currently produces events for the following services types:

- Lambda
- SNS
- SQS

39.6 Events consumed by this service

The S3 service does not consume events from other Handel services.

CHAPTER 40

S3 Static Site

This document contains information about the S3 Static Site service supported in Handel. This Handel service sets up an S3 bucket and CloudFront distribution for your static website.

Attention: This service requires you to have the external AWS CLI installed in order to use it. See the AWS documentation for help on installing it.

If you are running Handel inside CodePipeline, you should already have the AWS CLI pre-installed.

40.1 Service Limitations

40.1.1 No CORS Support

This service doesn't support configuring CORS support on the static site bucket. It just uses the default CORS configuration for S3 buckets:

- Origin: *
- Methods: GET
- Headers: Authorization

40.1.2 No Redirects Support

This service doesn't yet support redirects (i.e. 'www.mysite.com' to 'mysite.com') to your static site bucket.

40.2 Parameters

This service takes the following parameters:

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *s3staticsite* for this service type.
path_to_code	string	Yes		The path to the folder where your static website resides. This will be uploaded to your S3 static site bucket.
bucket_name	string	No	<appName>-<environmentName>-<serviceName>-<serviceType>	The name of the bucket to create. This name must be globally unique across all AWS accounts, so 'myBucket' will likely be taken. :)
versioning	string	No	disabled	Whether to enable versioning on the bucket. Allowed values: 'enabled', 'disabled'
index_document	string	No	index.html	The name of the file in S3 to serve as the index document.
error_document	string	No	error.html	The name of the file in S3 to serve as the error document.
cloudfront	*CloudFront Configuration*	No		Configuration for CloudFront. If not specified, CloudFront is not enabled.
tags	*Resource Tags*	No		Any tags you want to apply to your S3 bucket

40.2.1 CloudFront Configuration

The *cloudfront* section is defined by the following schema:

Parameter	Type	Required	Default	Description
https_certificate	string	No		The ID of an Amazon Certificate Manager certificate to use for this site
minimum_https_protocol	string	No	'TLSv1.2_2018'	The minimum allowed HTTPS protocol version. Valid values are listed in the Cloudfront API Docs.
dns_names	List<string>	No		The DNS names to use for the CloudFront distribution. See *DNS Records*.
price_class	string	No	all	one of *100*, *200*, or *all*. See CloudFront Pricing.
logging	enabled|disabled	No	enabled	Whether or not to log all calls to Cloudfront.
min_ttl	*TTL Values*	No	0	Minimum time to cache objects in CloudFront
max_ttl	*TTL Values*	No	1 year	Maximum time to cache objects in CloudFront
default_ttl	*TTL Values*	No	1 day	Default time to cache objects in CloudFront

TTL Values

min_ttl, *max_ttl*, and *default_ttl* control how often CloudFront will check the source bucket for updated objects. They are specified in seconds. In the interest of readability, Handel also offers some duration shortcuts:

Alias	Duration in seconds
second(s)	1
minute(s)	60
hour(s)	3600
day(s)	86400
year	31536000

So, writing this:

```
cloudfront_max_ttl: 2 days
```

is equivalent to:

```
cloudfront_max_ttl: 172800
```

40.3 Example Handel File

This Handel file shows an S3 Static Site service being configured:

```
version: 1

name: s3-static-website

environments:
  dev:
    site:
      type: s3staticsite
      path_to_code: ./_site/
      versioning: enabled
      index_document: index.html
      error_document: error.html
      cdn:
        price_class: all
        https_certificate: 6afbc85f-de0c-4ee9-b7d7-28b961eca135
      tags:
        mytag: myvalue
```

40.4 Depending on this service

The S3 Static Site service cannot be referenced as a dependency for another Handel service.

40.5 Events produced by this service

The S3 Static Site service does not produce events for other Handel services.

40.6 Events consumed by this service

The S3 Static Site service does not consume events from other Handel services.

CHAPTER 41

SES (Simple Email Service)

This document contains information about the SES service supported in Handel. This Handel service verifies an email address for use by your applications.

Note: This service does not currently support resource tagging.

41.1 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *ses* for this service type.
address	string	Yes		The email address your applications will use.

Note: When Handel attempts to verify an email address through SES, AWS will send an email to the address with a link to verify the address. Handel will not attempt to re-verify email addresses that have already been verified in the same AWS account or are in a pending state (SES allows 24 hours before a verification fails). It will still wire up the appropriate permissions to allow other Handel services to use successfully verified addresses.

Handel does not support verification of entire domains at this time.

Warning: To allow multiple applications to share an email address, Handel does not delete an SES identity upon deletion of the Handel SES service.

41.2 Example Handel File

This Handel file shows an SES service being configured:

```
version: 1

name: my-email-address

environments:
  dev:
    email:
      type: ses
      address: user@example.com
```

41.3 Depending on this service

This service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_EMAIL_ADDRESS	The email address available through SES
<SERVICE_NAME>_IDENTITY_ARN	The AWS ARN of the email identity

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

41.4 Events produced by this service

The SES service does not currently produce events for other Handel services.

41.5 Events consumed by this service

The SES service does not currently consume events from other Handel services.

CHAPTER 42

SNS (Simple Notification Service)

This document contains information about the SNS service supported in Handel. This Handel service provisions an SNS topic for use by your applications.

42.1 Service Limitations

Important: This service only offers limited tagging support. SNS Topics will not be tagged, but the Cloudformation stack used to create them will be. See *Tagging Unsupported Resources*.

42.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *sns* for this service type.
subscriptions	*Subscriptions*	No		An optional list of statically-defined subscriptions. You can also dynamically add subscriptions in your application code.
tags	*Resource Tags*	No		Tags to be applied to the Cloudformation stack which provisions this resource.

42.2.1 Subscriptions

The Subscription element is defined by the following schema:

```
subscriptions:
  - endpoint: <string>
    protocol: <http|https|email|email-json|sms>
```

See the SNS subscription documentation for full details on configuring endpoints and protocols.

Note: Protocols *sqs*, *application*, and *lambda* are supported through *Service Events*.

42.3 Example Handel File

This Handel file shows an SNS service being configured:

```
version: 1

name: my-sns-topic

environments:
  dev:
    topic:
      type: sns
      subscriptions:
        - endpoint: fake@example.com
          protocol: email
```

42.4 Example Handel File

This Handel file shows an SNS Topic as a dependency to a Lambda Function

```
version: 1

name: my-lambda-sns-example
environments:
  dev:
    function:
      type: lambda
      path_to_code: .
      handler: lambda_function.lambda_handler
      runtime: python3.6
      timeout: 180
      dependencies:
      - topic
    topic:
      type: sns
      subscriptions:
        - endpoint: fake@example.com
          protocol: email
```

42.5 Depending on this service

This service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_TOPIC_ARN	The AWS ARN of the created topic
<SERVICE_NAME>_TOPIC_NAME	The name of the created topic

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

42.6 Events produced by this service

The SNS service currently produces events for the following services types:

- Lambda
- SQS

42.7 Events consumed by this service

The SNS service currently consumes events for the following service types:

- CloudWatch Events
- S3

CHAPTER 43

SQS (Simple Queue Service)

This document contains information about the SQS service supported in Handel. This Handel service provisions an SQS queue for use by your applications.

43.1 Service Limitations

Important: This service only offers limited tagging support. SNS Topics will not be tagged, but the Cloudformation stack used to create them will be. See *Tagging Unsupported Resources*.

43.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *sqs* for this service type.
queue_type	string	No	regular	The type of queue to create. Allowed values are "regular" and "fifo".
delay_seconds	number	No	0	The amount of time the queue delays delivery of messages.
content_based_deduplication	boolean	No	false	Whether to enable content-based deduplication. This value only applies when the queue_type is "fifo".
max_message_size	number	No	262144	The max message size in bytes. Allowed values: 0 - 262144
message_retention_period	number	No	345600	The amount of time in seconds to retain messages. Allowed values: 60 - 1209600
receive_message_wait_time_seconds	number	No	0	The number of seconds ReceiveMessage will wait for messages to be available. Allowed values: 0-20. See Amazon SQS Long Polling for more information.
visibility_timeout	number	No	30	The amount of time a message will be unavailable after it is delivered from the queue. Allowed values: 0 - 43200
dead_letter_queue	*DeadLetterQueue*	No		If present, indicates that the queue will use a Dead-Letter Queue.
tags	*Resource Tags*	No		Tags to be applied to the Cloudformation stack which provisions this resource.

43.2.1 DeadLetterQueue

The *dead_letter_queue* section is defined by the following schema:

```
dead_letter_queue:
  max_receive_count: <number> # Optional.  Default: 3
  delay_seconds: <number> # Optional. Default: 0
  max_message_size: <number> # Optional. Default 1: queue max_message_size. Default␣
↪2: 262144
  message_retention_period: <number> # Optional. Default 1: queue message_retention_
↪period. Default 2: 345600
  receive_message_wait_time_seconds: <number> # Optional. Default 1: queue receive_
↪message_wait_time_seconds. Default 2: 0
  visibility_timeout: <number> # Optional. Default 1: queue visibility_timeout. ␣
↪Default 2: 30
```

If you want to use the default values, set *dead_letter_queue* to true:

```
dead_letter_queue: true
```

43.3 Example Handel Files

43.3.1 Simple Configuration

This Handel file shows a basic SQS service being configured:

```
version: 1

name: my-sqs-queue

environments:
  dev:
    queue:
      type: sqs
```

43.3.2 Dead-Letter Queue

This Handel file shows an SQS service being configured with a Dead-Letter Queue:

```
version: 1

name: my-sqs-queue

environments:
  dev:
    queue:
      type: sqs
      queue_type: fifo
      content_based_deduplication: true
      delay_seconds: 2
      max_message_size: 262140
      message_retention_period: 345601
      receive_message_wait_time_seconds: 3
      visibility_timeout: 40
      dead_letter_queue:
        max_receive_count: 5
        queue_type: fifo
        content_based_deduplication: true
        delay_seconds: 2
        max_message_size: 262140
        message_retention_period: 345601
        receive_message_wait_time_seconds: 4
        visibility_timeout: 40
```

43.3.3 Lambda Events

This Handel file shows an SQS service configured with events to Lambda enabled:

```
version: 1

name: my-sqs-queue

environments:
  dev:
    queue:
      type: sqs
      event_consumers:
      - service_name: function
        batch_size: 10
    function:
```

(continues on next page)

(continued from previous page)

```
      type: lambda
      path_to_code: .
      handler: index.handler
      runtime: nodejs8.10
```

43.4 Depending on this service

The SQS service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_QUEUE_NAME	The name of the created queue
<SERVICE_NAME>_QUEUE_URL	The HTTPS URL of the created queue
<SERVICE_NAME>_QUEUE_ARN	The AWS ARN of the created queue

If you have a Dead-Letter Queue, the SQS service also outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_DEAD_LETTER_QUEUE_NAME	The name of the created dead-letter queue
<SERVICE_NAME>_DEAD_LETTER_QUEUE_URL	The HTTPS URL of the created dead-letter queue
<SERVICE_NAME>_DEAD_LETTER_QUEUE_ARN	The AWS ARN of the created dead-letter queue

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

43.5 Events produced by this service

The SQS service produces events to the following service types:

- Lambda

You can configure events to Lambda using the *event_consumers* parameter in your SQS service:

```
event_consumers:
- service_name: <string> # Required.  The service name of the lambda function
  batch_size: <number> # Required. Allowed Values: 1-10
```

43.6 Events consumed by this service

The SQS service can currently consume events from the following Handel services:

- S3
- SNS

CHAPTER 44

Step Functions

This document contains information about the Step Functions service supported in Handel. This Handel service provisions Step Functions state machine resources to provide an application workflow.

44.1 Service Limitations

44.1.1 No Activities

This service does not yet support Step Functions activity resources. Task resources are limited to Lambda functions.

44.2 Parameters

Parameter	Type	Required	Default	Description
type	string	Yes		This must always be *stepfunctions* for this service type.
definition	*State Machine Definition*	Yes		Path to file containing state machine definition.

44.2.1 State Machine Definition

For the most part, the definition file you provide in the *definition* section is in Amazon States Language. Instead of providing an ARN in the 'Resource' field of a state, however, one should give the service name from the Handel file.

Note: For convenience, Handel supports both JSON and YAML formats for the definition file, where pure States Language is based on JSON alone.

A definition file could look something like this:

```
StartAt: FooState
States:
  FooState:
    Type: Task
    Resource: foo # service name
    Next: BarState
  BarState:
    Type: Task
    Resource: bar # service name
    End: true
```

44.3 Example Handel File

```
version: 1

name: my-state-machine

environments:
  prd:
    foo:
      type: lambda
      path_to_code: foo/
      handler: lambda_function.lambda_handler
      runtime: python3.6
    bar:
      type: lambda
      path_to_code: bar/
      handler: lambda_function.lambda_handler
      runtime: python3.6
    machine:
      type: step_functions
      definition: state_machine.yml # definition file
      dependencies:
      - foo
      - bar
```

44.4 Depending on this service

The Lambda service outputs the following environment variables:

Environment Variable	Description
<SERVICE_NAME>_STATE_MACHINE_NAME	The name of the created Step Functions state machine
<SERVICE_NAME>_STATE_MACHINE_ARN	The ARN of the created Step Functions state machine

See *Environment Variable Names* for information about how the service name is included in the environment variable name.

44.5 Events produced by this service

The Step Functions service does not produce events for other Handel services to consume.

44.6 Events consumed by this service

The Step Functions service does not consume events from other Handel services.

CHAPTER 45

Handel Deployment Logs

For internal use as well as an audit trail, Handel writes some information regarding the deployment and deletion of a Handel environment to a DynamoDB table named: *handel-deployment-logs*.

45.1 Log Entry Structure

After every deployment and every deletion for each environment, Handel will put an entry into the *handel-deployment-logs* DynamoDB table.

Field	Key	Type	Description
AppName	Partition Key	String	The application name being deployed/deleted
EnvAction	Sort Key	String	A combination of EnvironmentName, Lifecycle and timestamp
Lifecycle		String	"deploy" or "delete"
EnvironmentName		String	The environment that was deployed or deleted (i.e. "dev" or "prd")
DeploymentStartTime		Number	The timestamp in milliseconds (since the epoc) of when the deployment/deletion was initiated
DeploymentEndTime		Number	The timestamp in milliseconds (since the epoc) of when the deployment/deletion finished
DeploymentStatus		String	"success" or "failure"
DeploymentMessage		String	Success or failure message
ApplicationTags		JSON Object	A JSON representation of the application tags applied to each resource in the handel file
EnvironmentContents		JSON Object	A JSON representation of the environment's Handel services that were deployed or deleted

Here's an example deployment entry:

```
{
    "AppName": "test-app",
    "EnvAction": "dev:deploy:1536357426736",
    "Lifecycle": "deploy",
    "EnvironmentName": "dev",
    "DeploymentStartTime": 1536357268101,
    "DeploymentEndTime": 1536357426736,
    "DeploymentStatus": "success",
    "DeploymentMessage": "Success",
    "ApplicationTags": {
        "app": "test-app",
        "team": "The-Cool-Team"
    },
    "EnvironmentContents": {
        "my-lambda": {
            "type": "lambda",
            "path_to_code": ".",
            "handler": "index.handler",
            "runtime": "nodejs6.10"
            "dependencies": [
                "my-db"
            ]
        },
        "my-db": {
            "type": "mysql",
            "database_name": "test_db",
            "mysql_version": "5.6.27"
        }
    }
}
```

CHAPTER 46

Writing Extensions

This page contains information on how to write a custom Handel extension. You can use extensions to provide your own customized service types that retain the same automatic dependency wiring as the built-in Handel services.

Note: If you're looking for information on how to use a custom extension that someone else wrote, see the *Using Extensions* page.

46.1 Introduction

Handel is written in TypeScript on the Node.js platform. Therefore, implementing a Handel extension involves creating an NPM package.

Writing your extensions in TypeScript is highly recommended since the objects dealt with in the AWS world can be very large and complex, and Handel passes a lot of information around between service provisioners.

46.2 Creating an Extension

You can use the provided Yeoman generator to create a working extension skeleton with a single service. You can then use this skeleton to implement whatever you need in your extension.

First, install Yeoman and the generator:

```
npm install -g yo
npm install -g generator-handel-extension
```

Next, create a new directory and run the generator:

```
mkdir test-handel-extension
cd test-handel-extension
yo handel-extension
```

Answer the questions the generator asks:

```
Welcome to the handel-extension generator!
? Extension name
? Extension description
? Service type name
```

It will then create the output files in your directory:

```
Creating the initial files for the extension
  create package.json
identical .gitignore
  create README.md
  create tsconfig.json
  create tslint.json
  create src/extension.ts
  create src/service.ts
  create test/fake-account-config.ts
  create test/service-test.ts
```

46.2.1 Building the Extension

Now that you have your extension created, you can build it and run the unit tests:

```
npm install
npm run build
npm test
```

All of these commands should work successfully on the initial extension skeleton code.

46.2.2 Testing the Extension

Once you have your extension skeleton created and built properly, you can write a Handel file and run Handel to test the extension locally.

First, link your extension package so it is findable by Handel:

```
npm link
```

Next, create an example Handel file that will use your extension:

```
mkdir example
cd example
vim handel.yml
```

You can use something like the following as the contents of the Handel file:

```
version: 1

name: extension-test

extensions:
  test: test-handel-extension # NPM package name is of format <extensionName>-handel-
↪extensionj
```

(continues on next page)

(continued from previous page)

```
environments:
  dev:
    service:
      type: test::test # Service type that was specified is 'test'
```

The above handel file assumes that you chose *test* as your extension name and *test* as your service name when running the generator. If you specified something else you'll have to modify the contents of this file.

Finally, you can run Handel with the *–link-extensions* flag enabled to allow it to find your extension locally rather than from NPM:

```
handel deploy -c default-us-west-2 -e dev --link-extensions
```

46.2.3 Extension Support Package

If you look at the *package.json* file that was generated for your extension, you'll notice that it includes the *handel-extension-support* package as a dependency. This package contains useful functions that you can use when implementing the different phase types in your deployers.

For example, it contains a methods to easily do things like the following:

- Create a security group in the preDeploy phase.
- Bind a security group to another with ingress rules
- Create and wait for a CloudFormation template

You should look at the methods offered by that package, because they will likely save you time and effort when implementing your extension. See the package documentation for those details.

46.3 Extension Contract

Each Handel extension must expose a consistent interface that Handel can use to load and provision the service deployers contained inside it.

The following TypeScript interface defines the contract for an extension:

```
export interface Extension {
    loadHandelExtension(context: ExtensionContext): void | Promise<void>;
}
```

Your extension should use the passed-in ExtensionContext to add one or more service provisioners to it.

46.4 Service Provisioner Contract

A Handel extension is composed of one or more *dervice deployers*. Each service deployer must implement a particular contract consisting of one or more *phase types*. The Handel framework will invoke these implemented phase types at the appropriate time during deployment. Your job as an extension developer is to implement the phase types required for your service, and then Handel will take care of calling them at the right time and feeding them the correct data they need for deployment.

The following TypeScript interface defines the contract for a service deployer:

```
export interface ServiceDeployer {
    // ------------------------------------------------
    // Required metadata for the provisioner
    // ------------------------------------------------
    providedEventType: ServiceEventType | null; // The type of event type this␣
↪deployer provides (if any)
    producedEventsSupportedTypes: ServiceEventType[]; // The types of event types␣
↪that this deployer can produce to (if)
    producedDeployOutputTypes: DeployOutputType[]; // The types of deploy output␣
↪types this deployer produces to other deployers
    consumedDeployOutputTypes: DeployOutputType[]; // The types of deploy output␣
↪types this deployer can consume from other deployers
    supportsTagging: boolean; // If true, indicates that a deployer supports tagging␣
↪its resources. This is used to enforce tagging rules.

    // ------------------------------------------------
    // Phase types that hte provisioner supports
    // ------------------------------------------------
    /**
    * Checks the given service configuration in the user's Handel file for required␣
↪parameters and correctness.
    * This provides a fail-fast mechanism for configuration errors before deploy is␣
↪attempted.
    *
    * You should probably always implement this phase in every service deployer
    */
    check?(serviceContext: ServiceContext<ServiceConfig>,␣
↪dependenciesServiceContexts: Array<ServiceContext<ServiceConfig>>): string[];

    /**
    * Create resources needed for deployment that are also needed for dependency␣
↪wiring
    * with other services.
    *
    * Implement this phase if you'll be creating security groups for any of your␣
↪resources
    *
    * Example AWS services that would need to implement this phase include Beanstalk␣
↪and RDS.
    *
    * NOTE: If you implement preDeploy, you must implement getPreDeployContext as well
    */
    preDeploy?(serviceContext: ServiceContext<ServiceConfig>): Promise
↪<PreDeployContext>;

    /**
    * Get the PreDeploy context information without running preDeploy
    *
    * Return null if preDeploy has not been executed yet
    */
    getPreDeployContext?(serviceContext: ServiceContext<ServiceConfig>): Promise
↪<IPreDeployContext>;

    /**
    * Bind two resources from the preDeploy phase together by performing some wiring␣
↪action on them. An example
    * is to add an ingress rule from one security group onto another.
```

(continues on next page)

(continued from previous page)

```
    *
    * Bind is run from the perspective of the service being consumed, not the other␣
↪way around. In other words, it
    * is run on the dependency who is adding the ingress rule for the dependent␣
↪service.
    *
    * Implement this phase if you'll be creating resources that need to add ingress␣
↪rules for dependent services
    * to talk to them
    *
    * Example AWS services that would need to implement this phase include RDS and EFS
    */
    bind?(ownServiceContext: ServiceContext<ServiceConfig>, ownPreDeployContext:␣
↪IPreDeployContext, dependentOfServiceContext: ServiceContext<ServiceConfig>,␣
↪dependentOfPreDeployContext: IPreDeployContext): Promise<IBindContext>;

    /**
    * Deploy the resources contained in your service deployer.
    *
    * You are responsible for using the outputs in the dependenciesDeployContexts to␣
↪wire up this service
    * to those. For example, each one may return an IAM policiy that you should add␣
↪to whatever role is
    * created for your service.
    *
    * All this service's dependencies are guaranteed to be deployed before this phase␣
↪gets called
    */
    deploy?(ownServiceContext: ServiceContext<ServiceConfig>, ownPreDeployContext:␣
↪IPreDeployContext, dependenciesDeployContexts: IDeployContext[]): Promise
↪<IDeployContext>;

    /**
    * In this phase, this service should make any changes necessary to allow it to␣
↪consume events from the given source
    * For example, a Lambda consuming events from an SNS topic should add a Lambda␣
↪Function Permission to itself to allow
    * the SNS ARN to invoke it.
    *
    * This method will only be called if your service is listed as an event consumer␣
↪in another service's configuration.
    */
    consumeEvents?(ownServiceContext: ServiceContext<ServiceConfig>,␣
↪ownDeployContext: IDeployContext, eventConsumerConfig: ServiceEventConsumer,␣
↪producerServiceContext: ServiceContext<ServiceConfig>, producerDeployContext:␣
↪IDeployContext): Promise<IConsumeEventsContext>;

    /**
    * In this phase, this service should make any changes necessary to allow it to␣
↪produce events to the consumer service.
    * For example, an S3 bucket producing events to a Lambda should add the event␣
↪notifications to the S3 bucket for the
    * Lambda.
    *
    * This method will only be called if your service has an event_consumers element␣
↪in its configruation.
    */
```

(continues on next page)

(continued from previous page)

```
    produceEvents?(ownServiceContext: ServiceContext<ServiceConfig>,␣
↪ownDeployContext: IDeployContext, eventConsumerConfig: ServiceEventConsumer,␣
↪consumerServiceContext: ServiceContext<ServiceConfig>, consumerDeployContext:␣
↪IDeployContext): Promise<IProduceEventsContext>;

    /**
    * In this phase, the service should remove all resources created in the preDeploy␣
↪phase.
    *
    * Implment this phase if you implemented the preDeploy phase!
    */
    unPreDeploy?(ownServiceContext: ServiceContext<ServiceConfig>): Promise
↪<IUnPreDeployContext>;

    /**
    * In this phase, the service should remove all bindings on preDeploy resources.
    */
    unBind?(ownServiceContext: ServiceContext<ServiceConfig>): Promise<IUnBindContext>
↪;

    /**
    * In this phase, the service should delete resources created during the deploy␣
↪phase.
    *
    * Note that there are no 'unConsumeEvents' or 'unProduceEvents' phases. In most␣
↪cases, deleting the
    * service will automatically delete any event bindings the service itself has,␣
↪but in some cases this phase will
    * also need to manually remove event bindings. An example of this is CloudWatch␣
↪Events, which requires that
    * you remove all targets before you can delete the service.
    */
    unDeploy?(ownServiceContext: ServiceContext<ServiceConfig>): Promise
↪<IUnDeployContext>;
}
```

See the types in the *handel-extension-api* package for full details on the types passed as parameters to these phase type methods.

46.5 Handel Lifecycles

The above service deployer contract gives information about the different *kinds* of phase types, but not *when* they are invoked by the Handel framework.

The Handel tool supports multiple *lifecycles*. There are currently three lifecycles:

- Deploy - Deploys an application from a Handel file
- Delete - Deletes an environment in a Handel file
- Check - Checks the Handel file for errors

Each of these lifecycles runs through a pre-defined series of *phases*. The following sections explain the phase orders used by each lifecycle.

46.5.1 Deploy Lifecycle

The Deploy lifecycle executes the following phases in order:

1. Check
2. PreDeploy
3. Bind
4. Deploy
5. ConsumeEvents
6. ProduceEvents

46.5.2 Delete Lifecycle

The Delete lifecycle executes the following phases in order:

1. UnDeploy
2. UnBind
3. UnPreDeploy

46.5.3 Check Lifecycle

The Check lifecycle executes the following phases in order:

1. Check