# Handel Documentation

***Release 0.11.0***

**David Woodruff**

**Aug 21, 2017**

# Getting Started


1 Introduction 3

2 Handel vs. CloudFormation 5

3 Creating Your First Handel App 13

4 Handel File 17

5 Account Config File 19

6 Service Dependencies 21

7 Consuming Service Dependencies 23

8 Service Events 27

9 Accessing Application Secrets 29

10 Tagging 31

11 Deleting an Environment 33

12 Alexa Skill Kit 35

13 API Access 37

14 API Gateway 41

15 Beanstalk 45

16 CloudWatch Events 51

17 DynamoDB 55

18 ECS (Elastic Container Service) 61

19 EFS (Elastic File System) 69

20 IoT 71

21 Lambda 73

22 Memcached (ElastiCache) 77

23 MySQL (RDS) 81

24 PostgreSQL (RDS) 85

25 Redis (ElastiCache) 89

26 Route 53 Hosted Zone 93

27 S3 (Simple Storage Service) 97

28 S3 Static Site 101

29 SNS (Simple Notification Service) 105

30 SQS (Simple Queue Service) 107

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

CHAPTER 1

# Introduction

Handel is a library that will help you deploy your application to AWS. Configuring automated provisioning and deployment for your applications in AWS can be very difficult and time-consuming. Handel takes care of as many of the painful little details of your deployment as it can, so that you can focus on just configuring the services you want to use.

## Why does Handel exist?

An example will help illustrate the complexity of deploying AWS applications: If you want to use the EC2 Container Service (ECS) to run an application, you'll need to configure several resources. You'd need to provision and deploy at least the following resources:

- ECS Cluster
- Auto-Scaling Group
- Launch Configuration
- ECS Service
- Application Load Balancer (ALB)
- ALB Target Group
- ALB Listener

Once you've got those services provisioned, you'll need to wire them together securely with IAM roles and EC2 security groups. Doing this securely requires an in-depth knowledge of the IAM and VPC services.

After getting your ECS cluster running, you may decide you want to use other AWS services in that application. For example, if you want to use DynamoDB, you'll have to do the following:

- Figure out how to provision DynamoDB
- Wire DynamoDB into your ECS service with IAM permissions.

Other services such as EFS will have different patterns of how to wire it in to your ECS cluster, so you'll have to learn those eventually too.

All these steps contribute to an extremely steep learning curve when you want to go create a CloudFormation template that will securely deploy your application and the services on which it depends.

## How is Handel different than other deployment mechanisms?

Handel is an abstraction on top of CloudFormation. It does not replace CloudFormation, it actually uses CloudFormation under the hood to deploy your applications.

Handel provides the following benefits over vanilla CloudFormation:

- A much simpler interface to deploying an application. A 400-line CloudFormation template can be configured in more like 20-30 lines in a Handel file.
- Services are securely wired together for you with EC2 security groups.
- Services are securely wired together with IAM roles.
- Your application is injected with environment variables at run-time. These environment variables provide information about the location and configuration of consumed AWS services.

## What services are supported?

Handel does not support all AWS services. See the *Supported Services* section for information on which services you can use with Handel.

## How can I deploy an application with Handel?

See the *Creating Your First Handel App* page for a tutorial 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.

## 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
        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
        - 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
          - 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
    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

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
```

## 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
    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

# 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.

## Tutorial

This tutorial contains the following steps:

1. Write the app
2. Create your Handel file
3. Create the account config file
4. Deploy using Handel

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

### 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.

## 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 2016.09 v4.0.1 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.

## Create the account config file

Handel needs a base account configuration on which it can deploy your services. In particular, it needs to know about the VPC information for your account so it can know where to deploy things.

In order to provide this information, Handel requires that you give it an *Account Config File*, which contains this VPC information. This account config file is created once for your entire AWS account, and every app you deploy in that account can use this same file.

VPC setup and configuration is pretty painful, so for the purposes of this tutorial we're assuming you don't want to do this yourself. You have one of three options to generate this account config file:

1. If you're deploying in an account where someone else is already using Handel, that means they've already set up the VPCs and created the account config file. Just get the account config file from them.
2. If you're deploying in an account that doesn't use Handel yet, but already has a VPC configuration specified, get the person who set up the VPC to help you generate the account config file.
3. If you're deploying in an account that doesn't have any VPC stuff set up yet, you can use Handel-Quickstart to help you easily set up a reasonable VPC. This tool will auto-generate the account config file for you.

Once you've obtained your account config file using one of the three above methods, put it in a file called *account-config.yml* in your home directory.

**Danger:** The account-config file contains sensitive information such as VPC IDs, account IDs, and regions. **Do not** commit this file to your repository on GitHub or anywhere else that is publicly visible.

## Deploy using Handel

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 ~/account-config.yml -e dev -v 1
```

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

- The -c parameter specifies where your account config file is located
- The -e parameter is a comma-separated string list that specifies which environments from your Handel file you want to deploy
- The -v parameter is an arbitrary string specifying the current version being deployed. We're just using "1" in our example

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.

# Next Steps

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

## 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 the information you need to get started using Handel.

## Learn how to configure the different service types

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:

- Service features that aren't yet supported in Handel.
- Configuring 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).

## Set up a continuous delivery pipeline

Handel itself can run anywhere, but the best way to run Handel is inside a continuous delivery pipeline. AWS provides the CodePipeline service for continuous delivery pipelines. Handel provides a companion tool, called Handel-CodePipeline, that helps you easily create these pipelines running Handel for your deploy.

CHAPTER 4

# 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.

## 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>

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

## 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.

## 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 <environment_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.*

---

## Limits

The following limits exist on names in the deploy spec:

| 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 5

# Account Config File

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

- Your handel.yml file that contains your service specification
- An account configuration YAML file that contains account-level information for things such as VPCs, subnets, etc.

The account configuration file contains the low-level information that can be shared by all Handel apps deploying to a single account. This file is a bit more tricky to build than your Handel file, as it requires some knowledge of the network topology of your account.

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.

## Defining Your Own Account Config File

It's best if you can find someone with a good knowledge of VPCs to help define your account config file. If you don't have a person like that, see *Creating Your First Handel App* for a tutorial on setting up prerequisite resources and specifying your own account config file.

## Account Config File Specification

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

```
account_id: <aws account id>
region: <aws region>
vpc: <id for vpc in which to deploy compute resources>
public_subnets:
- <id for subnets in which to deploy public resources>
private_subnets:
- <id for subnets in which to deploy private resources>
```

```
data_subnets:
- <id for subnets in which to deploy data resources>
elasticache_subnet_group: <name of the cache subnet group to use>
ssh_bastion_sg: <id for the SSH bastion security group>
```

CHAPTER 6

# 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.

## 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 7

# 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:
      - queue
    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.

## Environment Variable Prefix

Every environment variable injected by Handel for service dependencies has a common prefix in the environment variable name.

This environment variable prefix is defined with the following structure:

```
<SERVICE_TYPE>_<APP_NAME>_<ENVIRONMENT_NAME>_<SERVICE_NAME>
```

These values come from the service dependency in your Handel file. In the above example, the referencing Beanstalk application would need to use the following values in that prefix:

```
service_type = "sqs"
app_name = "beanstalk-example"
environment_name = "dev"
service_name = "queue"
```

You can use the *Common Injected Environment Variables* to dynamically obtain values such as *environment_name* that will be different depending on which environment your code is currently running in.

---

**Note:** All Handel injected environment variables will be all upper-cased, with dashes converted to underscores. In the above example, the Beanstalk application would need to use the following prefix for the SQS queue:

```
SQS_BEANSTALK_EXAMPLE_DEV_QUEUE
```

Note that everything in the above prefix is upper-cased, and the app name "beanstalk-example" has been converted to to use underscores instead of dashes

---

## 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>.<service_name>
```

These values come from the service dependency in your Handel file. In the above example, the referencing Beanstalk application would need to use the following values in that prefix:

```
app_name = "beanstalk-example"
environment_name = "dev"
service_name = "queue"
```

You can use the *Common Injected Environment Variables* to dynamically obtain values such as *environment_name* that will be different depending on which environment your code is currently running in.

## 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_SERVICE_VERSION | This is the value of the version of the application being deployed. It is set to whatever the *-v* parameter was when Handel last deployed your application. |

CHAPTER 8

# 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.

## 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 9

# 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.
>
> Handel provides a different mechanism for passing secrets to your application, as explained in this document.

## 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 prefix. Handel wires up permissions for parameters with the following prefix:

```
<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:
      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 that start with "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 does not start with the proper prefix of *<appName>.<environmentName>* in the Handel file.

## 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.

---

## 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 docuemntation of the SDK you are using for examples.

CHAPTER 10

# 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.

## Handel Support for Tagging

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.

AWS services have limits on the total number of tags that may be applied to each service. Generally that limit is currently 50 tags.

See a service such as *EFS (Elastic File System)* for an example about how you can apply tags to Handel services.

## 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.

CHAPTER 11

# 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. You are responsible fo

To delete an environment, do the following:

Execute Handel's delete lifecycle, passing in the environment you want to delete:

```
# Note that you need to also pass in the account config file
handel delete -c ~/projects/byu/handel-account-configs/prd-swat-oit-byu.yml -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.
```

```
? 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 12

# 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.

## Parameters

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

## Example Handel Files

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

## 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.

## Events produced by this service

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

- Lambda

## Events consumed by this service

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

CHAPTER 13

# 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 exammple you would not be able to do things like start instances, create AMIs, etc.

## 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. |

### 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

## 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
```

## 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.

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 14

# 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.

## Service Limitations

### Only Proxy Service

This service will currently only provision an API that serves as a proxy for a single Lambda function. The API gateway has exactly one handler:

```
ANY on /{proxy+} # Routes all HTTP methods on any route to a single Lambda.
```

This means all your routing will have to be done inside the Lambda from the passed-in information.

If there is interest, there could be future support for API gateway specified by a Swagger document with some custom extensions that would allow you to tell what resources should be hosted by which lambdas.

### No Authorizer Lambdas

This service doesn't yet support specifying authorizer lambdas.

### No VPC Support

This service does not yet support running the Lambdas inside a VPC. It is easily added, but was not required by any apps thus far.

## Parameters

| Parameter | Type | Required | Default | Description |
|---|---|---|---|---|
| type | string | Yes | | This must always be *apigateway* for this service type. |
| path_to_code | string | Yes | | The path to the directory or artifact where your code resides. |
| lambda_runtime | string | Yes | | The Lambda runtime (such as nodejs6.10) to use for your handler function. |
| handler_function | string | Yes | | 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. |
| binary_media_types | array | No | | A sequence (array) of BinaryMediaType strings. *Note* The handel will do the '/' to '~1' character escaping for you. |
| description | string | No | Handel-created function | The configuration description of your Lambda function. |
| provisioned_memory | number | No | 128 | The amount of memory (in MB) to provision for the runtime. |
| function_timeout | string | No | 3 | The timeout to use for your Lambda function. Any functions that go over this timeout will be killed. |
| environment_variables | *EnvironmentVariables* | No | | A set of key/value pairs to set as environment variables on your API. |
| tags | *Tags* | No | | Any tags you want to apply to your Beanstalk environment |

### 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.

### Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## Example Handel File

This Handel file shows an API Gateway service being configured:

```
version: 1

name: my-apigateway-app

environments:
  dev:
    app:
      type: apigateway
      path_to_code: .
      lambda_runtime: nodejs6.10
      handler_function: index.handler
      provisioned_memory: 256
      function_timeout: 5
      environment_variables:
        MY_FIRST_VAR: my_first_value
        MY_SECOND_VAR: my_second_value
```

## Depending on this service

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

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 15

# 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.

## Service Limitations

### 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.

## 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 ECS service (if any) |
| environment_variables | *Environment-Variables* | No | | Any user-specified environment variables to inject in the application. |
| tags | *Tags* | No | | Any tags you want to apply to your Beanstalk environment |

## 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. |

### 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'
      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.

## 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.

## Routing

The Routing element is defined by the following schema:

```
routing:
  type: <http|https>
  https_certificate # 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.

## Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Attention:** Beanstalk tags may not be modified after you initially create the environment. Beanstalk has had a feature request open for years to modify tags on environments, but still doesn't support it.

If you try to modify your *tags* element after your environment is created, your CloudFormation stack will fail to update.

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

# Example Handel Files

## 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
```

## 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
            threshold: 30
            period: 60
```

### 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
```

## Depending on this service

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

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 16

# 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.

## Parameters

| Parameter | Type | Required | Default | Description |
|---|---|---|---|---|
| type | string | Yes | | This must always be *cloudwatchevents* 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 | string | No | | The list of event patterns on which to fire the event. In this field you just specify an Event Pattern in YAML syntax. |

## Example Handel Files

### 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"}'
```

### 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
      schedule: rate(1 minute)
      event_pattern:
        source:
        - aws.ec2
        detail-type:
        - EBS Volume Notification
        detail:
        event:
        - createVolume
      event_consumers:
      - service_name: function
```

## 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.

## Events produced by this service

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

- Lambda

## Events consumed by this service

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

CHAPTER 17

# DynamoDB

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

## Service Limitations

The following features are currently not supported:

- Local secondary indexes
- Global secondary indexes

## Parameters

| Parameter | Type | Required | Default | Description |
|---|---|---|---|---|
| type | string | Yes | | This must always be *dynamodb* for this service type. |
| 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 | 5 for read and write | The ProvisionedThroughput element details how much provisioned IOPS you want on your table for reads and writes. |
| 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 | *Tags* | No | | Any tags you want to apply to your Dynamo Table |

### 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>
```

### 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>
```

### 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>
  write_capacity_units: <number>
```

### 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>
```

### 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: # Required
  - <string>
  provisioned_throughput: # Optional
    read_capacity_units: <number> # Default: 1
    write_capacity_units: <number> # Default: 1
```

**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.

### Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## 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_capcity_units: 6
        write_capacity_units: 6
      tags:
        name: my-dynamodb-tag
```

## Depending on this service

The DynamoDB service outputs the following environment variables:

| Environment Variable | Description |
|---|---|
| <ENV_PREFIX>_TABLE_NAME | The name of the created DynamoDB table |
| <ENV_PREFIX>_TABLE_ARN | The ARN of the created DynamoDB table |

The <ENV_PREFIX> is a consistent prefix applied to all information injected for service dependencies. See *Environment Variable Prefix* for information about the structure of this prefix.

## 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
```

### 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.

## Events produced by this service

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

- Lambda

## Events consumed by this service

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

CHAPTER 18

# 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.

## Service Limitations

### 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.

### 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.

# 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. |
| dns | List<*DNS Records*> | No | | DNS records to create and associate with the load balancer for this task. |
| tags | *Tags* | No | | This section allows you to specify any tags you wish to apply to your ECS service. |

## 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
```

## 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*.

### 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
```

### 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.
```

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

### Tags

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

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## Example Handel Files

### Simplest Possible ECS Service

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

```
version: 1

name: my-ecs-app

environments:
  dev:
    webapp:
      type: ecs
      auto_scaling:
        min_tasks: 1
        max_tasks: 1
```

```
      containers:
      - name: mywebapp
```

## 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: /
```

## 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
      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
```

## 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
```

## 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
```

## Depending on this service

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

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 19

# 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.

## 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 | *Tags* | No | | Any tags you wish to apply to this EFS mount. |

### Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## Example Handel File

```
version: 1

name: my-efs-app
```

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

## Depending on this service

The EFS service outputs the following environment variables:

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

The <ENV_PREFIX> is a consistent prefix applied to all information injected for service dependencies. See *Environment Variable Prefix* for information about the structure of this prefix.

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 20

# 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.

## 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.

## Parameters

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

## 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
```

## 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.

## Events produced by this service

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

- Lambda

### 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. |

## Events consumed by this service

The IoT service cannot currently consume events from other services.

CHAPTER 21

# 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.

## Service Limitations

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

- Running Lambdas inside VPCs.
- Encrypting environment variables with KMS keys

# 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 | string | No | 128 | The amount of memory to allocate for your function |
| timeout | string | No | 3 | The timeout in seconds for your function. Max 300 |
| environment_variables | *EnvironmentVariables* | No | | Any environment variables you want to inject into your code. |
| tags | *Tags* | No | | Any tags you want to apply to your Lambda |

## 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.

## Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

# Example Handel File

```
version: 1

name: my-lambda

environments:
```

```
dev:
  webapp:
    type: lambda
    path_to_code: .
    handler: index.handler
    runtime: nodejs6.10
    environment_variables:
      MY_ENV: myEnvValue
    tags:
      mytag: mytagvalue
```

## 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.

## Depending on this service

The Lambda service cannot currently be consumed by any other services. It is intended as an event consumer for other services such as SNS.

## Events produced by this service

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

## Events consumed by this service

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

- SNS

CHAPTER 22

# Memcached (ElastiCache)

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

## Service Limitations

### No Scheduled Maintenance Window Configuration

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

### No Snapshot Window Configuration

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

### No Restoration From Snapshot

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

## 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 | *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.**

### Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## 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
```

## Depending on this service

The Memcached service outputs the following environment variables:

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

The <ENV_PREFIX> is a consistent prefix applied to all information injected for service dependencies. See *Environment Variable Prefix* for information about the structure of this prefix.

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 23

# MySQL (RDS)

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

## Service Limitations

### 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.

### 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.

## Parameters

| Parameter | Type | Required | Default | Description |
|---|---|---|---|---|
| type | string | Yes | | This must always be *mysql* for this service type. |
| 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. |
| mysql_version | string | no | 5.6.27 | The version of MySQL you wish to run. See MySQL on Amazon RDS for the list of supported versions. |
| db_username | string | no | handel | The username for the user that will be created in your database. Your password will be automatically generated and securely stored in the EC2 Parameter Store for you to access. |
| storage_type | string | no | standard | The type of storage to use, whether magnetic or SSD. Allowed values: 'standard', 'gp2'. |
| 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 | *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.

### Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## 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
  db_username: mydb
  storage_type: standard
  db_parameters:
    autocommit: 1
  tags:
    mytag: myvalue
```

## Depending on this service

The MySQL service outputs the following environment variables:

| Environment Variable | Description |
|---|---|
| <ENV_PREFIX>_ADDRESS | The DNS name of the MySQL database address. |
| <ENV_PREFIX>_PORT | The port on which the MySQL instance is listening. |
| <ENV_PREFIX>_USERNAME | The username you can use to access the database. |
| <ENV_PREFIX>_DATABASE_NAME | The name of the database in your MySQL instance. |

The <ENV_PREFIX> is a consistent prefix applied to all information injected for service dependencies. See *Environment Variable Prefix* for information about the structure of this prefix.

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

| Parameter Name | Description |
|---|---|
| <parameter_prefix>.db_password | The password for your database user. |

The <parameter_prefix> 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.

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 24

# PostgreSQL (RDS)

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

## Service Limitations

### 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.

### 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.

## 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. |
| 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. |
| postgres_version: | string | no | 9.6.2 | The version of PostgreSQL you wish to run. See PostgreSQL on Amazon RDS for the list of supported versions. |
| db_username | string | no | handel | The username for the user that will be created in your database. Your password will be automatically generated and securely stored in the EC2 Parameter Store for you to access. |
| storage_type | string | no | standard | The type of storage to use, whether magnetic or SSD. Allowed values: 'standard', 'gp2'. |
| 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 | *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.

### Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## 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
  db_username: mydb
  storage_type: standard
  db_parameters:
    authentication_timeout: 600
  tags:
    mytag: myvalue
```

## Depending on this service

The PostgreSQL service outputs the following environment variables:

| Environment Variable | Description |
|---|---|
| <ENV_PREFIX>_ADDRESS | The DNS name of the PostgreSQL database address. |
| <ENV_PREFIX>_PORT | The port on which the PostgreSQL instance is listening. |
| <ENV_PREFIX>_USERNAME | The username you can use to access the database. |
| <ENV_PREFIX>_DATABASE_NAME | The name of the database in your PostgreSQL instance. |

The <ENV_PREFIX> is a consistent prefix applied to all information injected for service dependencies. See *Environment Variable Prefix* for information about the structure of this prefix.

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

| Parameter Name | Description |
|---|---|
| <parameter_prefix>.db_password | The password for your database user. |

The <parameter_prefix> 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.

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 25

# Redis (ElastiCache)

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

## Service Limitations

### 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.

### No Restoration From Snapshot

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

## 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. |
| 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. |
| redis_version | string | Yes | | The version of Redis to run. See Comparing Redis Versions for a list of available versions. |
| 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 | *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**

### Tags

The Tags element is defined by the following schema:

```
tags:
  <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## Example Handel File

```
version: 1

name: my-redis-cluster

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

## Depending on this service

The Redis service outputs the following environment variables:

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

The <ENV_PREFIX> is a consistent prefix applied to all information injected for service dependencies. See *Environment Variable Prefix* for information about the structure of this prefix.

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 26

# 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.

## Service Limitations

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

- Domain Name Registration

## 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.

## 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 | *Tags* | No | | Any tags you want to apply to your Hosted Zone |

### Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## 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
```

## Depending on this service

This service outputs the following environment variables:

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

The <ENV_PREFIX> is a consistent prefix applied to all information injected for service dependencies. See *Environment Variable Prefix* for information about the structure of this prefix.

### DNS Records

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

- Beanstalk
- ECS

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

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 27

# 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.

## 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 lifecycle
- Bucket logging
- Cross-region replication

## Parameters

This service takes the following parameters:

| Parameter | Type | Required | Default | Description |
|---|---|---|---|---|
| type | string | Yes | | This must always be *s3* for this service type. |
| 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. :) |
| bucket_acl | string | No | disabled | **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*. |
| tags | *Tags* | No | | Any tags you want to apply to your S3 bucket |

### Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## Example Handel File

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
```

## Depending on this service

This service outputs the following environment variables:

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

The <ENV_PREFIX> is a consistent prefix applied to all information injected for service dependencies. See *Environment Variable Prefix* for information about the structure of this prefix.

## Events produced by this service

The S3 service does not currently produce events for other Handel services. Support is planned to be added in the future.

## Events consumed by this service

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

CHAPTER 28

# S3 Static Site

This document contains information about the S3 Static Site service supported in Handel. This Handel service sets up an S3 bucket 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.

## Service Limitations

### No CORS Support

This service doesn't yet support configuring CORS support on the static site bucket.

### No Redirects Support

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

## 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. |
| tags | *Tags* | No | | Any tags you want to apply to your S3 bucket |

### Tags

The Tags element is defined by the following schema:

```
tags:
 <your_tag_name>: <your_tag_value>
```

**Note:** Handel automatically applies some tags for you. See *Default Tags* for information about these tags.

## 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
      tags:
        mytag: myvalue
```

## Depending on this service

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

## Events produced by this service

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

## Events consumed by this service

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

CHAPTER 29

# 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.

## 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. |

### 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*.

## Example Handel File

This Handel file shows an SQS service being configured:

```
version: 1

name: my-sns-topic

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

## Depending on this service

This service outputs the following environment variables:

| Environment Variable | Description |
|---|---|
| <ENV_PREFIX>_TOPIC_ARN | The AWS ARN of the created topic |
| <ENV_PREFIX>_TOPIC_NAME | The name of the created topic |

The <ENV_PREFIX> is a consistent prefix applied to all information injected for service dependencies. See *Environment Variable Prefix* for information about the structure of this prefix.

## Events produced by this service

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

- SQS
- Lambda

## Events consumed by this service

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

CHAPTER 30

# 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.

## 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 | string | 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. |

### DeadLetterQueue

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

```
dead_letter_queue:
  max_receive_count: <number> # Optional.  Default: 3
  queue_type: <string> # Optional. Default: regular
  content_based_deduplication: <boolean> # Optional. Default: false
  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
```

## Example Handel Files

This Handel file shows a basic SQS service being configured:

```
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
```

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
```

## Depending on this service

The SQS service outputs the following environment variables:

| Environment Variable | Description |
|---|---|
| <ENV_PREFIX>_QUEUE_NAME | The name of the created queue |
| <ENV_PREFIX>_QUEUE_URL | The HTTPS URL of the created queue |
| <ENV_PREFIX>_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 |
|---|---|
| <ENV_PREFIX>_DEAD_LETTER_QUEUE_NAME | The name of the created dead-letter queue |
| <ENV_PREFIX>_DEAD_LETTER_QUEUE_URL | The HTTPS URL of the created dead-letter queue |
| <ENV_PREFIX>_DEAD_LETTER_QUEUE_ARN | The AWS ARN of the created dead-letter queue |

The <ENV_PREFIX> is a consistent prefix applied to all information injected for service dependencies. See *Environment Variable Prefix* for information about the structure of this prefix.

## Events produced by this service

The SQS service does not produce events for other Handel services.

## Events consumed by this service

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

- SNS