# How to Setup an Nginx Reverse Proxy using Amazon Lightsail Containers

In this guide, you'll learn how to configure a Flask web server behind an Nginx reverse proxy using Lightsail containers. The Nginx reverse proxy accepts web requests on port 80 and forwards them to the Flask web server on port 5000. The Flask web server fulfills the requests and return the response to Nginx. A Lightsail container service will be created to host both the Nginx and the Flask containers. A public endpoint will be created to allow external access to the Nginx server.

To get started, you'll need an [AWS account](https://portal.aws.amazon.com/billing/signup) and must install [Docker](https://docs.docker.com/engine/install/), [Docker compose](https://docs.docker.com/compose/install/), the [AWS Command Line Interface (CLI) tool](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html) and the [Lightsail Control (lightsailctl) plugin](https://lightsail.aws.amazon.com/ls/docs/en_us/articles/amazon-lightsail-install-software) on your system. Follow the provided links if you don't have some of those.

## 1. Get the source code

1. The source code to accompany this guide can be found in this repository. To get started, clone the GitHub respository locally.

bash git clone https://github.com/awsgeek/lightsail-containers-nginx.git

1. Change to the project directory bash cd lightsail-containers-nginx

## 2. The Flask application

The Flask application contains a single hello\_world() function that is triggered when the route / is requested. When run, this application binds to all IPs on the system ("0.0.0.0") and listens on port 5000 (this is the default Flask port).

The source for the Flask application, app.py, is shown below.

```python from flask import Flask app = Flask(**name**)

@app.route("/") def hello\_world(): return "Hello, World!"

if **name** == "**main**": app.run(host="0.0.0.0", port=5000) ```

The Dockerfile for the Flask application uses a Python alpine image to minimize container image size. The command to run when the container starts is the same as if run from the command line: python app.py

``` # Set base image (host OS) FROM python:3.8-alpine

# By default, listen on port 5000 EXPOSE 5000/tcp

# Set the working directory in the container WORKDIR /app

# Copy the dependencies file to the working directory COPY requirements.txt .

# Install any dependencies RUN pip install -r requirements.txt

# Copy the content of the local src directory to the working directory COPY app.py .

# Specify the command to run on container start CMD ["python", "./app.py"] ```

## 3. Build the Flask container

Complete the following steps to build the Flask applicaiton container on your local system.

1. Build the container using Docker. Execute the following command from the project directory: docker build -t flask-container ./flask This command builds a container using the Dockerfile in the current directory and tags the container flask-container.
2. Once the container build is done, test the Flask application locally by running the container: docker run -p 5000:5000 flask-container
3. The Flask app will run in the container and will be exposed to your local system on port 5000. Browse to http://localhost:5000 or use curl from the command line and you will see Hello, World!

``` curl localhost:5000

Hello, World! ```

## 4. The Nginx reverse proxy

The Nginx reverse proxy forwards all requests to the Flask application on port 5000. Configuring Nginx to forward reqests reuqires a simple configuration file, nginx.conf:

``` events {}

http {

upstream flask {  
 server ${FLASK\_HOST}:${FLASK\_PORT};  
 }  
  
 # a simple reverse-proxy  
 server {  
  
 listen 80 default\_server;  
  
 location / {  
 # pass requests to the Flask host  
 proxy\_pass http://flask;  
 }  
 }

} ```

This configuration forwards all requests to the upstream Flask server. The hostname and port of the Flask server are provided as environmental variables when the containers are run.

The Dockerfile for the Nginx reverse proxy uses the master Nginx alpine image and simply copies the nginx.conf configuration file to the appropriate directory.

FROM nginx:1.19-alpine COPY ./nginx.conf /etc/nginx/templates/nginx.conf.template ## 5. Build the Nginx container

Complete the following steps to build the Nginx reverse proxy container on your local system.

1. Build the container using Docker. Execute the following command from the project directory: docker build -t nginx-container ./nginx This command builds a container using the Dockerfile in the current directory and tags the container nginx-container.
2. Once the container build is done, test the Nginx proxy and Flask application locally by running the container: docker-compose up --build
3. Both the Flask application and Nginx reverse proxy containers will be run. The Nginx server listens for requests on port 80 and forwards them to the Flask application. Browse to http://localhost or use curl from the command line and you will see Hello, World!.

``` curl localhost

Hello, World! ```

## 6. Create a container service

Complete the following steps to create the Lightsail container service and then push your local container images to the new container service.

1. Create a Lightsail container service with the [create-container-service](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lightsail/create-container-service.html) command.

aws lightsail create-container-service --service-name sample-service \ --power small \ --scale 1 The power and scale parameters specify the capacity of the container service. For the purposes of this guide, little capacity is required.

The output of the create-container-service command indicates the state of the new service is PENDING. { "containerService": { "containerServiceName": "sample-service", ... "state": "PENDING",

Use the [get-container-services](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lightsail/get-container-services.html) command to monitor the state of the container as it is being created.

aws lightsail get-container-services --service-name sample-service

Wait until the container service state changes to ACTIVE before continuing to the next step. Your container service should become active after a few minutes.

1. Push the Flask application container to Lightsail with the [push-container-image](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lightsail/push-container-image.html) command. ``` aws lightsail push-container-image --service-name sample-service  
   --label flask-container  
   --image flask-container

...

Refer to this image as ":sample-service.flask-container.X" in deployments. Note: theXinsample-service.flask-container.X``` will be a numeric value. If this is the first time you’ve pushed an image to your container service, this number will be 1. You will need this number in the next step.

1. Push the Nginx reverse proxy container to Lightsail with the [push-container-image](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lightsail/push-container-image.html) command. ``` aws lightsail push-container-image --service-name sample-service  
   --label nginx-container  
   --image nginx-container

...

Refer to this image as ":sample-service.nginx-container.Y" in deployments. Note: theYin:sample-service.nginx-container.Y``` will be a numeric value. If this is the first time you’ve pushed an image to your container service, this number will be 1. You will need this number in the next step.

## 7. Deploy the containers

Complete the following steps to create deployment and public endpoint configuration JSON files, and then deploy your container images to your container service.

1. Create a new file, containers.json. Edit the file and add the following. Replace the X in :sample-service.flask-container.X and the Y in :sample-service.nginx-container.Y with the numeric values from the previous step. Save the file. ``` { "sample-nginx": { "image": ":sample-service.nginx-container.Y", "command": [], "ports": { "80": "HTTP" }, "environment": { "NGINX\_ENVSUBST\_OUTPUT\_DIR": "/etc/nginx", "FLASK\_HOST": "localhost", "FLASK\_PORT": "5000" } }, "sample-flask": { "image": ":sample-service.flask-container.X", "ports": { "5000": "HTTP" } } }

Thecontainers.json``` file describes the settings of the containers that will be launched on the container service. In this instance, the containers.json file describes both the Nginx and Flask containers, the images the containers will use and the ports the containers will expose. In addition, environmental variables that specify the Flask host and port are provided. At runtime Nginx will replace the placeholder values in the nginx.conf file with the actual values provided here.

1. Create a new file, public-endpoint.json. Edit the file and add the following. Save the file. { "containerName": "sample-nginx", "containerPort": 80 } The public-endpoint.json file describes the settings of the public endpoint for the container service. In this instance, the public-endpoint.json file indicates the Nginx container will expose port 80. Public endpoint settings are only required for services that require public access.
2. Deploy the containers to the container service with the AWS CLI using the [create-container-service-deployment](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lightsail/create-container-service-deployment.html) command.

aws lightsail create-container-service-deployment --service-name sample-service \ --containers file://containers.json \ --public-endpoint file://public-endpoint.json The output of the create-container-service-deployment command indicates that the state of the container service is now DEPLOYING. { "containerServices": [{ "containerServiceName": "sample-service", ... "state": "DEPLOYING", Use the [get-container-services](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lightsail/get-container-services.html) command to monitor the state of the container until it changes to RUNNING before continuing to the next step.

aws lightsail get-container-services --service-name sample-service

The get-container-service command also returns the endpoint URL for container service.

{ "containerServices": [{ "containerServiceName": "sample-service", ... "state": "RUNNING", ... "url": "https://sample-service... After the container service state changes to RUNNING, navigate to this URL in your browser to verify your container service is running properly. Your browser output should show "Hello, World!" as before.

Congratulations. You have successfully deployed a containerized Nginx reverse proxy and Flask application using Amazon Lightsail containers.

## Cleanup

Complete the following steps to the Lightsail container service that you created as part of this tutorial.

To cleanup and delete Lightsail resources, use the [delete-container-service](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lightsail/delete-container-service.html) command.

aws lightsail delete-container-service --service-name sample-service

The delete-container-service removes the container service, any associated container deployments, and container images.

## Additional Resources

The source code for this guide and this documentation is located in this [GitHub repository](https://github.com/AwsGeek/lightsail-containers-nginxq)