

Developer Guide

EchoMark API v2.0.3

Introduction.....	2
Why is EchoMark Useful?.....	2
Core Concepts	3
Installation	4
Packaging.....	5
Structure of API Zip File.....	5
EchoMark Deployment	10
Advanced Configuration	25
Marking	26
Marking Options.....	26
Marking Documents	30
Marking Static Workflow	35
Investigations	35
The meaning of a result.....	36
For Marked Documents	37
/detect/ endpoint.....	37

Introduction

This guide provides basic developer documentation for users of the EchoMark API in a “low dependency” configuration. This configuration is ideal for power users that want direct access to the EchoMark core algorithms without any frills.

Why is EchoMark Useful?

Forensic watermarking is a digital security technique used to embed imperceptible, yet identifiable, information into digital content such as videos, images, audio, or documents. Unlike visible watermarks, forensic watermarks are designed to be covert and resilient, remaining intact even after compression, resizing, or format changes. This embedded data typically contains unique identifiers that can trace the origin or distribution path of the content. The process is particularly valuable in environments where intellectual property and confidential data are at risk of unauthorized use or disclosure.

Many traditional forensic watermarks are vulnerable to the “analog hole”. Digital content must ultimately be rendered as visual or audio output—on screens, speakers, or printed pages—for users to consume it. At this point, the data becomes vulnerable to capture, as protections no longer apply to the analog form. This allows individuals to bypass digital safeguards by photographing screens, recording audio through speakers, or manually copying displayed information. For example, a DRM system with watermarking system may successfully block a user from transferring a confidential document to a USB device or sending it via email, but it cannot prevent that same user from taking a photo of the screen with a smartphone. The analog hole thus represents a fundamental challenge to information security because no digital control can fully prevent the unauthorized capture of data once it is displayed or played.

EchoMark specializes in visual content protection that embeds invisible marks into the content itself that often survives the analog-hole providing content creators additional tools to investigate such information leak vectors.

Core Concepts

First, we will review a few core concepts in the EchoMark API.

- **Content to mark or “original content”.** This refers to your content before watermarking. In v2.0 of the API, we support content in the form of images, pdfs, and html.
- **Recipient.** The identity of one intended recipient of forensically watermarked content. Recipients must be uniquely identified by some string.
- **Marked Copy.** A forensically watermarked version of the original content individualized to a particular recipient.
- **Artifact.** A reproduction of a marked copy copying its content and potentially changing format/media. An example of an artifact is a screenshot of a sensitive document posted to social media, i.e., an unauthorized disclosure.
- **Investigation.** The process of resolving an artifact to a marked copy and, thus, an intended recipient.

What is in the API (and what is not)?

This version of the API contains the core algorithms and excludes the following “business logic”:

1. User authentication and identity management

2. Original content repository
3. Caching and content-delivery optimizations

This version of the API assumes that either the above three points are implemented by you or they are not essential for what you are working on. Now, what are some scenarios what are some of the strengths of the API?

1. **Ease of installation.** You can deploy a standalone EchoMark API server within 30 minutes on the cloud or locally.
2. **Full Data Control.** You fully manage the deployment and all of the resources used. This is ideal for integrations where you may not want to share your data in a multi-user SaaS environment. Furthermore, you can launch this version of the API without any IT support.
3. **Smallest Building Block.** We can't anticipate every application you may want to build. Optimizations that make sense in our multi-user SaaS application, may not make sense for your use-case. The light API is the most flexible building block for forensic watermarking applications.
4. **Algorithms Research.** Power users might be interested in testing or benchmarking EchoMark's algorithms to understand how they work on your own data. The light API is ideal for this kind of evaluation.

Basic REST API Structure

The developer interacts with the API with HTTP endpoints. For marking tasks, a POST request can be issued to the appropriate endpoint with the original content, a set of marking options, and the intended recipient's identity. The request returns a response after marking is completed with the marked content as a byte stream in a multipart form message. Marking tasks are intended to be blocking and fast.

Investigation tasks may take longer so the workflow is slightly different. For these tasks, a POST request sends the appropriate content such as the artifact and potentially a copy of the original content to the API server. This request queues an asynchronous task to run in the background to complete the investigation. The developer's post request returns quickly with a call-back pointer to the investigation. There is further an interface to render the results of an investigation as a standalone html page to visualize the results.

Installation

The EchoMark API is distributed as a Docker container. Docker is a platform that allows developers to package applications and their dependencies into lightweight, portable

containers that run consistently across different environments. It simplifies deployment by using isolated containers that share the host OS kernel, ensuring efficient resource usage and reproducibility.

Packaging

The EchoMark API is packaged as a Zip archive file that contains the docker container as well as deployment scripts to help you get up and running. The main thing that you need to note is that you have a package for the right chip architecture (this will be clear in the filename).

From the EchoMark team, you'll receive a file that looks something like this:

`echomark-apiserver.2.0.0.amd64.63cb4a0d3e3f.zip`

Your EchoMark account manager will work with you to find the best way to deliver this file securely.

Versioning

`echomark-apiserver.2.0.0.amd64.63cb4a0d3e3f.zip`

Each container file will be tagged with a version identifier, e.g., `2.0.0`. There are some key developer guarantees around these versions and backwards compatibility.

Minor Version (0) Minor version changes are guaranteed to be fully backwards compatible. We will increment the minor version for changes that will not functionally affect the end-user such as security updates to dependencies.

Semi-Minor Version (0) Semi-minor version changes are guaranteed to be fully backwards compatible from a data processing perspective. All watermarked copies are inter-operable between semi-minor versions.

Major Version (2) Major version changes have no guarantees of backward compatibility.

Tag(63cb4a0d3e3f) Finally, there is a tag that identifies a hash of the particular image that your organization received. This is useful as an internal versioning tool if EchoMark customizes your deployment in any way.

Structure of API Zip File

Unzipping this file, you will see the following directory structure:

```
*.sh # launch and deployment script examples

*.yaml # cloud formation templates that help with aws installation

docker-compose/ # contains editable configuration such as ssl certificates ports, bearer
tokens

resources/ # auto-deploy scripts run on the launch in aws

images/ # the docker containers
```

Images Directory

EchoMark provides the raw container images that can be deployed manually as well. In the `./images`` directory, you'll see two tar files. These are serialized docker containers for the EchoMark API server ``apiserver.tar`` and the EchoMark admin server ``adminserver.tar``.

- The EchoMark API server serves the REST endpoints for the API. This does the actual work. By default this serves a HTTP rest API at port 5344.
- The EchoMark Admin server hosts a lightweight web interface for testing and investigations. By default this serves an HTTPS admin interface at port 5345 and a proxy for the API at port 5345 that ssl secures all API communication.

The API server can be run without the adminserver, if your hosting environment has web content delivery tools that replicate similar functionality.

API Server Environment Variables

Configuration for the API Server are expressed as UNIX environment variables that can be passed to the container on start. Example configuration can be found in ``docker-compose/conf/admin.conf``:

```
# Role of the container (e.g., "admin", "mark", "detect")
# Default: admin
CONTAINER_ROLE=admin

# Port to run the application on
# Default: 5344
PORT=5344

# CORS domains allowed (e.g., echomark.acme.org)
# Default: allow all
CORS_ALLOWED_ORIGINS=

# Path to the TLS certificate file (if using HTTPS)
# Default: not set
CERTIFICATE=
```

```
# Whether to skip authentication (use with caution)
# Accepts: True or False
# Default: False
SKIP_AUTH=False

# Path to the AWS Certificate Bundle
# Default: not set
AWS_CERTIFICATE_BUNDLE=/certs/aws

# Path to the private key for the TLS certificate
# Default: not set
PRIVATE_KEY=

# Type of metadata storage to use (e.g., "none", "sqlite", "postgres")
# Default: none
METADATA_STORAGE_TYPE=none

# Path to the SQLite database file (only used if METADATA_STORAGE_TYPE=sqlite)
# Default: /data/storage/metadata/sqlite.db
SQLITE_DATABASE_FILE=/data/storage/metadata/sqlite.db

# Hostname or IP address of the database server (used for non-SQLite databases)
# Default: not set
DATABASE_HOST=

# Port number for the database server
# Default: not set
DATABASE_PORT=

# Username for connecting to the database
# Default: not set
DATABASE_USER=

# Password for the database user
# Default: not set
DATABASE_PASSWORD=

# Number of threads to use for investigations
# Default: 2
INVESTIGATION_THREADS=2

# Path to store investigation results
# Default: /data/storage/results/
INVESTIGATION_RESULT_STORAGE=/data/storage/results/

# Enable verbose logging
# Accepts: True or False
# Default: False
VERBOSE=False
```

To launch a container with such configuration, one can run the following commands:

```
$ docker load -i "images/apiserver.tar" # get <image name from
output>
```

```
$ docker run --env-file docker-compose/conf/admin.conf <image_name>
```

There are two directories that may be useful to serve as read/write volumes to ensure that data persists across container runs.

- /certs this is the location for mTLS certificates for API Server interactions with cloud services.
- /data this is where local storage happens for the SQLITE metadata store and pending investigations.

```
$ docker load -i "images/apiserver.tar" # get <image name from output>

$ docker run -v local_path_to_certs:/certs -v local_path_to_data:/data --env-file docker-compose/conf/admin.conf <image_name>
```

For many deployments, the API server is sufficient for use. However, you may want to include the admin server for ease of testing.

Docker-Compose

Docker Compose is a tool that allows developers to define and manage multi-container Docker applications using a single YAML file, typically named `docker-compose.yml`. Instead of running individual `docker run` commands for each container, Docker Compose enables you to configure all services, networks, and volumes in one file and launch them with a single command (`docker compose up`). This simplifies workflows, especially for applications with multiple interconnected components such as a web server, database, and cache. It also supports features like service dependencies, environment variables, and persistent storage, making it easier to reproduce development and testing environments consistently.

There is a `docker-compose/docker-compose.yml` file that launches the API server and admin server with a default configuration. This configuration does a few things:

1. Runs the apiserver with REST endpoints exposed at port 5344

2. Runs the adminserver with a reverse proxy at 5345 with some static html pages hosted off that port as well.
3. From the archive file it mounts local paths as volumes with relevant data such as static html pages and certificates.
4. At the cloud provider level, one can set up a firewall to only allow global traffic through 5345 with limited internal traffic (e.g., just the local network or localhost) at 5344 for testing/heart beats.

```
# docker compose for the disa api server
version: '3.8'
services:
  adminserver:
    image: nginx:latest # Replace with your actual image ID or name
    container_name: adminserver
    ports:
      - "5345:443"
    networks:
      my_app_network:
        ipv4_address: 172.20.0.3
    volumes:
      - ./html:/usr/share/nginx/html:rw
      - ./conf:/tmp/conf:ro
    command: sh -c "cp /tmp/conf/cert.* /usr/share/nginx/ && cp /tmp/conf/admin.conf /usr/share/nginx/html/static/admin.conf && cp /tmp/conf/nginx.conf /etc/nginx/nginx.conf && cp /tmp/conf/config.js /usr/share/nginx/html/static/config.js && nginx -g 'daemon off;'"
    restart: unless-stopped

  apiserver:
    container_name: apiserver
    image: apiserver:latest # Replace with your actual image ID or name
    ports:
      - "5344:5344"
    networks:
      my_app_network:
        ipv4_address: 172.20.0.2
    env_file:
      - conf/admin.conf
    volumes:
      - /mnt/db:/data
      - ./conf/aws-certs/rds:/certs/db
    restart: unless-stopped
networks:
  my_app_network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/24
          gateway: 172.20.0.1
```

Similarly, the admin server has configuration that can be modified. This is standard NGINX configuration and found in `docker-compose/conf/nginx.conf`.

Standalone Deployment of Admin Server

The admin server is implemented as a standalone html/javascript application. This can be deployed without docker-compose. The files are located in `docker-compose/html` and can be served by any webserver. Modifying `docker-compose/conf/config.js` can point your admin server to any API endpoint that you host.

EchoMark Deployment

Eventually, you will build your own EchoMark deployment infrastructure but to get started quickly, we have provided example scripts in a number of important deployment scenarios. To be precise about where things happen, we will use the following terms:

- **Hosting Server.** Place where EchoMark API will be deployed.
- **Developer Device.** The device from which you are deploying to the hosting server.

Scenario 1. Installation on a Generic Hosted Server

*You have a pre-provisioned server with a *nix operating system, and a firewall rule that exposes port 5345 to the desired network.*

These instructions are largely agnostic to the cloud provider and tools available to you.

Prerequisites.

1. Install docker and docker compose on the target **hosting server**. Please select the appropriate linux distribution family that you are working with.
 - a. <https://docs.docker.com/engine/install/ubuntu/>
 - b. <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/install-docker.html>
 - c. <https://docs.docker.com/engine/install/rhel/>
 - d. Docker compose can be installed:
<https://docs.docker.com/compose/install/linux/>
2. Transfer the zip file to your **hosting server** and unzip. For example, you can use scp to transfer the data.

```
unzip echomark-apiserver.2.0.0.amd64.63cb4a0d3e3f.zip
cd echomark-apiserver.2.0.0.amd64.63cb4a0d3e3f
```

3. Update the configuration in docker-compose/conf as desired. The documentation above will help you understand what to change.

4. Run ``import_from_filesystem.sh`` on the **hosting server** to start both the admin server and the api server.

```
./import_from_filesystem.sh
```

5. In this deployment scheme, updates are safe. Let's say that you received an updated archive. You can simply rerun the steps to redeploy.

```
unzip echomark-apiserver.2.0.0.amd64.77cb4a0d0e39.zip
cd echomark-apiserver.2.0.0.amd64.77cb4a0d0e39
./import_from_filesystem.sh
```

Scenario 2. Installation on an AWS Hosted Server via S3

*You have a pre-provisioned server with a *nix operating system hosted in AWS, and a firewall rule that exposes port 5345 to the desired network.*

These instructions are more specific to AWS than above as they demonstrate preferred methods for file transfer and versioning. This transfer method is preferred for the following reasons:

- You may not have a public ip or dns to perform the transfer via a tool like SCP
 - You want to persistently store old versions of the EchoMark API and easily switch between versions.
1. Create an S3 bucket that will store all of your EchoMark data. You should have read/write/list privileges on this bucket AND the target **hosting server** should have those privileges as well either through an **Instance Profile** (https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2_instance-profiles.html) or via Access Keys. We're going to call ours "echomark-api".
 2. Install docker and docker compose on the target **hosting server**. Please select the appropriate linux distribution family that you are working with.
 - a. <https://docs.docker.com/engine/install/ubuntu/>
 - b. <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/install-docker.html>
 - c. <https://docs.docker.com/engine/install/rhel/>

- d. Docker compose can be installed:

<https://docs.docker.com/compose/install/linux/>

3. Install awscli on the target **hosting server**. Configure Access Keys if needed.

- a. <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>
- b. <https://docs.aws.amazon.com/cli/v1/userguide/cli-chap-configure.html>
- c. Test to see if you can access your aws s3 bucket from the server

```
$ aws s3 ls
2025-10-29 20:21:25 echomark-api
```

4. Install AWS CLI on **your developer device**. Configure Access Keys if needed.

- a. <https://docs.aws.amazon.com/cli/v1/userguide/cli-chap-configure.html>
- b. Test to see if you can access your aws s3 bucket from the server


```
$ aws s3 ls
2025-10-29 20:21:25 echomark-api
```

5. Download the archive to **your developer device** and unzip.

```
unzip echomark-apiserver.2.0.0.amd64.63cb4a0d3e3f.zip
cd echomark-apiserver.2.0.0.amd64.63cb4a0d3e3f
```

6. Update the configuration in docker-compose/conf as desired on your **developer device**. The documentation above will help you understand what to change.

7. Transfer the configured API container to S3. Run the following command on your **developer device**, replace “echomark-api” with whatever your bucket name is:

```
$ ./store_images_aws_s3.sh echomark-api
[EchoMark] Creating versioned subfolders in S3...
...
[EchoMark] Uploading local folders to S3...
...
 [EchoMark] Upload complete!
```

8. Import the containers on your **hosting server**. On your **developer device**, you will see an import script ``import_from_s3.sh``. Copy the contents of that script to your **hosting server**. E.g., via scp, ssm, or even cut-and-paste. You can run the import script as follows:

```
./import_from_s3.sh echomark-api 2.0.2.amd64.63cb4a0d3e3f  
# ./import_from_s3.sh <bucket name> <api version tag>
```

Scenario 3. Automated Installation in AWS

You have an AWS account with administrator privileges and will use deployment scripts to automatically provision a server, database, and security configuration.

If you are using AWS, we provide example templates to automatically deploy the infrastructure needed to host the API. We provide AWS installation scripts to automatically launch an API server with the appropriate networking so you can get started right away. These scripts are meant to be extensible so they serve as a starting point for a effective and secure API server deployment.

DISA STIGs (Defense Information Systems Agency Security Technical Implementation Guides) are standardized security configuration guidelines used by the U.S. Department of Defense to harden systems and ensure they meet strict cybersecurity requirements. SELinux (Security-Enhanced Linux) is a Linux kernel security module that enforces mandatory access control (MAC) policies to restrict what processes can access which resources. SELinux is often a key component in meeting DISA STIG requirements for Linux systems, as it provides fine-grained control over system interactions, helping to reduce the attack surface and enforce least privilege. By design the deployment scripts deploy a server host that is compliant with these rules.

Checklist to proceed:

- ☐ We assume that you have an AWS account and have basic familiarity with the AWS CLI
- ☐ We assume that you have an AWS role that has the appropriate permissions to create and launch new instances, security groups, and databases.

- ❑ Our scripts launch a hardened host by default from a third-party vendor, purchase a subscription if you wish to use the scripts unmodified

<https://aws.amazon.com/marketplace/pp/prodview-7jmrkmj4ev7w2>

Step 0. AWS Configuration and Setup

First, make sure you have configured an AWS profile that has the appropriate permissions to create and launch new instances.

Use `aws configure` to make sure your access key, secret key, and region are set.

```
$ export AWS_PROFILE=echomark-test
$ aws configure
AWS Access Key ID [*****UY4P]:
AWS Secret Access Key [*****L0oA]:
Default region name [us-west-1]:
Default output format [None]:
```

Before we can run any scripts there are a few network pre-requisites are usually specific to your environment.

First, get the VPC that you want to deploy to. This is often the "default" VPC, but can also be a custom one. Talk to your networking team on where you should be deploying EchoMark if you don't know. The following command can be used to see the VPCs available in your region.

```
$ aws ec2 describe-vpcs
{
  "Vpcs": [
    {
      "OwnerId": "890444671368",
      "InstanceTenancy": "default",
      "CidrBlockAssociationSet": [
        {
          "AssociationId": "vpc-cidr-assoc-03cb4fd912a151678",
          "CidrBlock": "10.0.0.0/24",
          "CidrBlockState": {
            "State": "associated"
          }
        }
      ],
      "IsDefault": false,
      "VpcId": "vpc-0520033e054b7abbe",
```

```

        "State": "available",
        "CidrBlock": "10.0.0.0/24",
        "DhcpOptionsId": "dopt-061b59aedfbc615d9"
    },
    {
        "OwnerId": "890444671368",
        "InstanceTenancy": "default",
        "CidrBlockAssociationSet": [
            {
                "AssociationId": "vpc-cidr-assoc-
0d0718fad7d10da0f",
                "CidrBlock": "172.31.0.0/16",
                "CidrBlockState": {
                    "State": "associated"
                }
            }
        ],
        "IsDefault": true,
        "VpcId": "vpc-08e69388ff8523409",
        "State": "available",
        "CidrBlock": "172.31.0.0/16",
        "DhcpOptionsId": "dopt-061b59aedfbc615d9"
    }
]
}

```

You should be able to find a vpc identifier (VPC_ID) along the lines of **vpc-08e69388ff8523409**.

Next, in this VPC you should identify the subnets that you will use. For deployments with a database, you need to specify TWO subnets. RDS Multi-AZ deployments are designed for high availability and automatic failover. These AZs must be in separate subnets, which is why you need at least two subnets. Even if you don't enable Mult-AZ deployment, you still need to specify two subnets. Again, this is a question for your networking team if you cannot create subnets on your own. You can extract subnets from a particular vpc with a command as follows replacing XXXX with your vpc id:

```

# aws ec2 describe-subnets --filters "Name=vpc-id,Values=vpc-XXXXXX"
$ aws ec2 describe-subnets --filters "Name=vpc-id,Values=vpc-08e69388ff8523409"
{
  "Subnets": [
    {
      "AvailabilityZoneId": "usw1-az3",

```

```
"MapCustomerOwnedIpOnLaunch": false,
"OwnerId": "890444671368",
"AssignIpv6AddressOnCreation": false,
"Ipv6CidrBlockAssociationSet": [],
"SubnetArn": "arn:aws:ec2:us-west-1:890444671368:subnet/subnet-
06fb37ef782be82f1",
"EnableDns64": false,
"Ipv6Native": false,
"PrivateDnsNameOptionsOnLaunch": {
  "HostnameType": "ip-name",
  "EnableResourceNameDnsARecord": false,
  "EnableResourceNameDnsAAAARecord": false
},
"SubnetId": "subnet-06fb37ef782be82f1",
"State": "available",
"VpcId": "vpc-08e69388ff8523409",
"CidrBlock": "172.31.0.0/20",
"AvailableIpAddressCount": 4089,
"AvailabilityZone": "us-west-1a",
"DefaultForAz": true,
"MapPublicIpOnLaunch": true
},
{
  "AvailabilityZoneId": "usw1-az1",
  "MapCustomerOwnedIpOnLaunch": false,
  "OwnerId": "890444671368",
  "AssignIpv6AddressOnCreation": false,
  "Ipv6CidrBlockAssociationSet": [],
  "SubnetArn": "arn:aws:ec2:us-west-1:890444671368:subnet/subnet-
036a2c7adac12cf8d",
  "EnableDns64": false,
  "Ipv6Native": false,
  "PrivateDnsNameOptionsOnLaunch": {
    "HostnameType": "ip-name",
    "EnableResourceNameDnsARecord": false,
    "EnableResourceNameDnsAAAARecord": false
  },
  "SubnetId": "subnet-036a2c7adac12cf8d",
  "State": "available",
  "VpcId": "vpc-08e69388ff8523409",
  "CidrBlock": "172.31.16.0/20",
  "AvailableIpAddressCount": 4090,
  "AvailabilityZone": "us-west-1b",
  "DefaultForAz": true,
```



```
    "MapPublicIpOnLaunch": true
  }
]
}
```

The result is that we need two subnet identifiers, one for the primary az and one for the replica. These should look like this `subnet-06fb37ef782be82f1/ subnet-036a2c7adac12cf8d` and we'll call them SUBNET1 and SUBNET2.

Checklist to proceed:

- ☐ VPC ID. The VPC to deploy EchoMark
- ☐ SUBNET1. The primary availability zone on which the API server will reside
- ☐ SUBNET2. The failover zone.

Unpacking EchoMark API Server

You'll get a self-contained archive file that looks something like this, unzip it:

```
$ unzip echomark-apiserver-aws-2.0.0.amd64.b73a67dc608b.zip
$ cd echomark-apiserver-aws-2.0.0.amd64.b73a67dc608b
```

Step 1. Instance Connection Method

Next, we need to determine how to connect to the AWS instance that will host your API.

Option 1. I already have an AWS Key Pair

The simplest way to administer an API server is using a named key-pair. In the first connection option, you can use an existing Key Pair to connect. Make sure that you have access to the this key pair.

```
$ aws ec2 describe-key-pairs
{
  "KeyPairs": [
    {
      "KeyId": "key-058936b54aa9b29f8",
      "KeyType": "rsa",
      "Tags": [],
      "CreateTime": "2025-10-30T01:54:54.385000+00:00",
      "KeyName": "echomark-deploy",
      "KeyFingerprint": "08:32:9b:4a:19:48:60:7c:98:05:60:1a:35:4d:b0:de:a5:aa:3e:e3"
```

```
    },  
    {  
      "KeyId": "key-08938057be4d27e71",  
      "KeyType": "rsa",  
      "Tags": [],  
      "CreateTime": "2025-08-12T17:10:37.911000+00:00",  
      "KeyName": "default",  
      "KeyFingerprint": "fc:fa:3e:f0:01:1d:a2:ae:fe:38:77:96:43:de:bc:f8:24:3f:7f:dc"  
    }  
  ]  
}
```

Option 2. I want to create a new key pair.

Alternatively, you can create a new key-pair. This is an AWS managed public-private key pair that will allow us to talk to your API server instance for deployment and update.

You will need the following IAM roles to proceed.

- "ec2:DescribeKeyPairs"
- "ec2:CreateTags",
- "ec2:DescribeTags"
- "ec2:CreateKeyPair"

The basic syntax of the script is:

```
./create_keypair.sh <name>
```

If you don't provide a name echomark-api-kp will be used. Running the script will create a new file <name>.pem in your current directory that is your private key. Please keep this safe as you will need this for future updating.

If successful, you will see a message like this:

```
$ ./create_aws_keypair.sh echomark-new-deploy  
[EchoMark] AWS region is configured: us-west-1
```

[EchoMark] Checking if key pair 'echomark-new-deploy' exists...

[EchoMark] Creating key pair 'echomark-new-deploy'...

[EchoMark] Key pair created successfully.

✅ [EchoMark] This is a private key to manage your API deployment, please keep it safe: echomark-new-deploy.pem

This is a private key to manage your API deployment, please keep it safe: `echomark-api-kp.pem`. This private key is what you will use to update your software in the future.

Checklist to proceed.

- ☐ Successfully created a keypair

Option 3. I'll manage my own connections to the server

If you don't want to use a key pair, you can skip this step and leverage alternatives such as:

1. AWS SSM
2. AWS Instance Connect Endpoints

[Optional] Step 2. Deploy an RDS Metadata Database

The next step is to launch a database which will store the metadata needed to run investigations. This step is optional in two scenarios. First, if you are simply testing out EchoMark, the API server will default to an SQLite Database hosted in the server container itself. Next, some deployments run in a "storageless" configuration where you manage the metadata yourself.

Note you need the following IAM roles to proceed.

- "cloudformation:CreateStack",
- "cloudformation:UpdateStack",
- "cloudformation:DescribeStacks",
- "cloudformation>DeleteStack",
- "cloudformation:GetTemplate",
- "cloudformation:ValidateTemplate"
- "rds:CreateDBInstance",

- "rds:DeleteDBInstance",
- "rds:DescribeDBInstances",
- "rds:CreateDBSubnetGroup",
- "rds>DeleteDBSubnetGroup",
- "rds:DescribeDBSubnetGroups"
- "ec2:CreateSecurityGroup",
- "ec2>DeleteSecurityGroup",
- "ec2:AuthorizeSecurityGroupIngress",
- "ec2:AuthorizeSecurityGroupEgress",
- "ec2:RevokeSecurityGroupIngress",
- "ec2:RevokeSecurityGroupEgress",
- "ec2:DescribeSecurityGroups",
- "ec2:DescribeSubnets",
- "ec2:DescribeVpcs"

With all the networking information, we can launch an rds instance. The stack name can be whatever you want, it's for your own internal tracking purposes:

```
./create_rds_instance.sh [STACK_NAME] [VPC_ID] [SUBNET1] [SUBNET2]
```

Note database settings can be changed in apiserver-rds.yaml. Running the script successfully should get you:

```
./create_rds_instance.sh apiserver vpc-0520033e054b7abbe subnet-
0c06a09b44a2fc3dd subnet-087b1b054bf2e0690
[EchoMark] AWS region is configured: us-west-1

[EchoMark] Deploying CloudFormation stack: apiserver using template apiserver-
rds.yaml.

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - apiserver-db
[EchoMark]Instance Creation Complete!
Edit conf/admin.conf with the following settings (the password is randomly chosen):
[
  {
    "OutputKey": "DBPassword",
```

```
[
  "OutputValue": "6VD1w2OG~jEj",
  "Description": "The password for the RDS instance"
},
{
  "OutputKey": "DBHost",
  "OutputValue": "apiserver-db.chcceyq44627.us-west-1.rds.amazonaws.com",
  "Description": "The endpoint of the created RDS instance"
},
{
  "OutputKey": "DBPort",
  "OutputValue": "5432",
  "Description": "The port of the RDS instance"
},
{
  "OutputKey": "DBUsername",
  "OutputValue": "echomark",
  "Description": "The username for the RDS instance"
}
]
```

Configuring the database settings

After running the RDS installation script. You need to update docker-compose/conf/admin.conf with the results of this script.

- METADATA_STORAGE_TYPE. Set this to be "postgres".
- DBHost (set DATABASE_HOST) This is the DNS name of the RDS instance. If the instance is in a VPC, this resolves to an internal IP address when accessed from within the same VPC.
- DBUsername (set DATABASE_USER) This is the username for the RDS instance. We default it to "echomark".
- DBPassword (set DATABASE_PASSWORD) This is the password for the created user "echomark" with a randomly generated password. Change in production if needed.
- DBPort (set DATABASE_PORT). Set this to 5432 for postgres.

Note these settings and update `docker-compose/conf/admin.conf` appropriately:

```
...
# Type of metadata storage to use (e.g., "none", "sqlite", "postgres")
# Default: none
METADATA_STORAGE_TYPE=postgres
```

```
# Path to the SQLite database file (only used if METADATA_STORAGE_TYPE=sqlite)
# Default: /data/storage/metadata/sqlite.db
SQLITE_DATABASE_FILE=/data/sqlite.db

# Hostname or IP address of the database server (used for non-SQLite databases)
# Default: not set
DATABASE_HOST=echomarkdbinstance.chcceyq44627.us-west-1.rds.amazonaws.com

# Port number for the database server
# Default: not set
DATABASE_PORT=5432

# Username for connecting to the database
# Default: not set
DATABASE_USER=echomark

# Password for the database user
# Default: not set
DATABASE_PASSWORD=6VD1w2OG~]Ej
...
```

More ways to configure the database are described in the “advanced” configuration section.

Step 3. Setting Up Data Transfer

Next, we will set up a way to get your EchoMark API container onto your server.

Option 1. Use AWS S3

Our preferred method for data transfer is to use an AWS S3 bucket to store your API Server. This bucket provides a persistent store for historical versions of the EchoMark API. Note using S3 with the following scripts will require IAM automation to setup an instance profile.

1. Create an S3 bucket that will store all of your EchoMark data.
2. Make sure you have access to the S3 bucket.

```
$ aws s3 ls
2025-10-29 20:21:25 echomark-api
```

```
$ # ./store_images_aws_s3.sh <bucket name>
```

```
$ ./store_images_aws_s3.sh echomark-api  
[EchoMark] Creating versioned subfolders in S3... ...  
[EchoMark] Uploading local folders to S3... ...  
✓ [EchoMark] Upload complete!
```

Option 2. Use Alternative

There is an alternatives if you don't want to use S3. The primary reason to not do this would be to avoid IAM automation.

1. If you have public dns or ip address access to your server, you can use our SCP based launch scripts using a key pair.

Step 4. Launching A Host Instance

Next, we will deploy an EchoMark API server. The following IAM roles are needed to proceed:

- "ec2:RunInstances",
- "ec2:TerminateInstances",
- "ec2:CreateSecurityGroup",
- "ec2:AuthorizeSecurityGroupIngress",
- "ec2:AuthorizeSecurityGroupEgress",
- "ec2:RevokeSecurityGroupIngress",
- "ec2:RevokeSecurityGroupEgress",
- "ec2:Describe*",
- "ec2>DeleteSecurityGroup",
- "ec2:CreateTags"

To launch an EC2 instance via S3:

```
$/create_aws_instance_via_s3.sh [STACK_NAME] [VPC_ID] [SUBNET1] [BUCKET NAME]  
[Optional...ssh-key]
```

Then, after you should see:

```
./create_aws_instance_via_s3.sh apiserver-new vpc-08e69388ff8523409 subnet-06fb37ef782be82f1 echomark-api echomark-new-deploy
```

...

```
✓ [EchoMark]Instance Creation Complete @ ec2-54-176-63-189.us-west-1.compute.amazonaws.com[EchoMark]
```

To launch an EC2 instance via scp instead of S3.

```
./create_aws_instance_via_scp.sh [STACK_NAME] [VPC_ID] [SUBNET1] [Optional...ssh-key]
```

```
$ ./create_aws_instance_via_scp.sh apiserver-new-2 vpc-08e69388ff8523409 subnet-06fb37ef782be82f1 echomark-new-deploy
```

```
✓ [EchoMark]Instance Creation Complete @ [EchoMark]
```

Re-Deploying/Updating EchoMark

Suppose you want to update an EchoMark API server with new configuration or a code update. Get a new API server archive file:

```
$ unzip echomark-apiserver-aws-2.0.0.amd64.b73a67dc608b.zip
$ cd echomark-apiserver-aws-2.0.0.amd64.b73a67dc608b
```

Configure `docker-compose/conf` to your liking based on the instructions above.

Store this current archive to S3 after configuration and run the update script.

```
$ ./store_images_aws_s3.sh BUCKET_NAME

$ ./update_aws_instance_via_s3.sh STACK_NAME BUCKET_NAME SSH_KEY_PAIR
```

If you would like to use scp instead.


```
$ ./update_aws_instance_via_scp.sh STACK_NAME SSH_KEY_PAIR
```

Advanced Configuration

The steps above provide a quick start guide to launching the API. However, not every deployment will look like that. Next, we will overview the structure of your downloaded API archive to understand what you can modify. Obviously, we can't be exhaustive but you'll get an idea of where to begin. Do reach out to support@echomark.com for any assistance.

Deploying to custom AWS hosts

You may not want to use our DISA STIG ami and use your own. EchoMark makes it easy to configure the host image. The installation scripts automatically copy everything in the `resources/` folder to the host machine. Line 34-36 can be modified in the `create_disa_instance.sh` script to execute any install script you would like.

```
#Rocky Linux Specific Config, change this if you are porting this script to ubuntu for
example
KEY_USER="rocky" # Replace with appropriate user, e.g., rocky or ubuntu, depending on
AMI_REMOTE_SCRIPT="resources/install_rocky9.sh" # this is the launch script that
installs the system
AMI_TO_USE="NONE" # set this if you would like to use a custom AMI
```

For example, we provide an example `install_ubuntu.sh` that can be used instead with an ubuntu ami.

The following external dependencies are needed.

- unzip
- dnf-plugins-core
- docker
- awscli
- docker-compose

Production Considerations

The main thing to configure is the database service. The EchoMark apiserver requires some form of persistent metadata storage. For debugging and testing you can use a local sqlite file. However, in production, we recommend using a relational database such as mysql or postgres sql. This database should have a database named **apiserver** and a user account that has write access to all tables in that database.

It is advisable to run multiple apiserver containers to scale up to the needs of your organization and host them behind a load balancer. Accordingly, each container can be assigned a "role" through environment variable configuration. Roles limit the functionality of the container so they can be aligned with your organizations security policies.

- "mark" The container will only perform watermarking tasks
- "detect" The container will only perform watermarking tasks and watermarking investigations
- "admin" The container can perform both mark and detect tasks, but also administrative tasks on the metadata server

Marking

Marking is the process of embedding forensic watermarks into original content.

Marking Options

EchoMark believes in a defense-in-depth strategy to forensic watermarking so there is no single watermarking algorithm that is applied to a given document. Instead, the developer is given a set of options that control where and how watermarks are placed. Adding more watermarks allows for more robust investigations, at the cost of slower marking time and increased chance for perceptible issues in the marked content. Mark options do have subtly different interpretations across formats (pdf, image, html) due to the nature of how content is represented. This section will give an overview of those differences and illustrate the options presented to the developer.

For images, html webpages, and pdf documents, mark options are specified in a JSON structure that nests each of the algorithm options above.

Name	Type	Description
recipient	string	An external identifier that uniquely identifies as recipient, i.e., disambiguates name collisions.
recipient_name	string	A human readable

		name that identifies the recipient (can be public).
image_watermarking_enabled	boolean	Enables invisible image watermarking on the asset or on embedded image files (e.g., in a pdf).
image_watermarking_options	ImageWatermarkingOptions	A json dictionary describing algorithm options for image watermarking.
text_watermarking_enabled	boolean	Enables invisible text watermarking that moves textual elements natively.
text_watermarking_options	TextWatermarkingOptions	A json dictionary describing options for native watermarking the text layer.
visible_watermarking_enabled	boolean	Enables a visible watermark with the

		specified settings.
visible_watermarking_options	VisibleWatermarkingOptions	A json dictionary describing options for a visible watermark.

Text Watermarking Options

Affects: PDF, HTML

Text watermarking involves embedding watermarks into textual content of each of the formats. This involves subtly adjusting the positioning of textual data in the document. Note that this technique only applies to content that is explicitly text such as PDF text and HTML text. It will not adjust text that happens to be in an image or a vector graphic that happens to look like a letter. This option has an effect for PDF and HTML marking, but will not have an effect for Image marking.

Name	Type	Description
rendering_intent	Options: {“screen”, “download”, “print”}	How the pdf is intended to be displayed. “Screen” will rasterize text if necessary, “download” will keep original file quality, and print will adjust the viewport such that all visible content is in the printable area. T

Image Watermarking Options

Affects: PDF, Image

Image watermarking involves embedding a forensic watermarking into image content whether directly as image content or as embedded within another file type, such as a PDF. In image watermarking, there are two methods that we can use:

- Luma. Our most robust image watermark is a good general-purpose watermark for scans, text-heavy content, tables, and photographs. It is robust to leaks via printout or low-quality photo. This mark works by applying subtle dilations to various parts of

the image, meaning that it can sometimes become noticeable, especially if your image contains repeating geometric patterns or strong diagonal lines.

- **Chroma.** Our most imperceptible watermark is best for full-color imagery. It can protect colorful photography and artwork without introducing any noticeable changes for viewers. It works well for protecting against leaks via screenshot or high-quality photo, but this method is not robust for leaks via printout or low-quality photo. Since it works by subtly altering color tones in your image, it offers no protection for black-and-white imagery or solid colored imagery such as backgrounds and single-color shapes.

Name	Type	Description
algorithm	Options: {“auto”, “chromatic”, “none”}	The algorithm used for image watermarking. Auto automatically selects a sensible default using luma when appropriate, chromatic only changes colors using the chroma algorithm, none does no watermarking, text watermarks in a way that preserves readability of rasterized text. The default value is “auto”
color_distortion_limit	number	The maximum color change introduced by the algorithm in 8-bit color space along each color channel.
spatial_distortion_limit	number	The maximum spatial distortion introduced by the algorithm in pixels. Basically, the maximum spatial movement allowed for a single pixel.

Visible Watermarking Options

Visible watermarks add visible text to your content, such as “Do Not Distribute”. This text can be customized in terms of content, color, opacity, and positioning. Furthermore, there is a macro “{{recipient}}” which can be used to dynamically insert the recipient’s name into the watermark. Note “{{recipient}}” inserts the “recipient_name” field described later in this document. The font used for watermarking is OpenSans which is distributed with the API.

Name	Type	Description
------	------	-------------

text	string	The text displayed in a visible watermark. The substring {{recipient}} can be used to insert a real world recipient name dynamically.
position	Options:{"footnote", "center"}	The position of the visible watermark. footnote places the watermark at the bottom of the document, overlay places the watermark diagonally over the content of the document.
color	string	The hexadecimal code for the color of the text.
opacity	number	A number between 0 (fully invisible) and 1 (fully opaque).

Example

For example, the following settings apply all three marking techniques for a recipient named Alice Bar.

```
{
  "recipient": "exployeeid_dwwhijtoarkbmkkz",
  "recipient_name": "Alice Bar",
  "image_watermarking_enabled": true,
  "image_watermarking_options": {
    "algorithm": "auto",
    "color_distortion_limit": 3,
    "spatial_distortion_limit": 5
  },
  "text_watermarking_enabled": true,
  "text_watermarking_options": {
    "rendering_intent": "screen"
  },
  "visible_watermarking_enabled": true,
  "visible_watermarking_options": {
    "text": "Personalized for {{recipient}}",
    "position": "footnote",
    "color": "000000",
    "opacity": 1
  }
}
```

Marking Documents

A document is any markable content that can be persistent and archived. In other words, for the duration of your use of EchoMark you need to be able to retrieve and present the original content during investigations. For document content, EchoMark recreates all of the marked copies during the investigation. This allows for a highly robust investigation of any potential artifact. However, recreation of these marked copies requires that the exact original content is provided to the system.

The API server maintains a list of all recipients for whom it has ever been requested to mark this original document, and also that EchoMark knows how to associate this original with each of those recipients if presented with exactly the same original used to generate those copies.

All of the static marking endpoints accept multipart form data as input with mark options and a file payload.

```
Content-Type: multipart/form-data; boundary=----FormBoundary7MA4YWxkTrZu0gW
-----FormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="original; filename="secure.jpg"
Content-Type: image/jpeg

[Binary data of the JPEG file]
-----FormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="mark_options"
Content-Type: application/json

{
  "recipient": "employeeid_dwwhijtoarkbmkz",
  "recipient_name": "Alice Bar",
  "image_watermarking_enabled": true,
  "image_watermarking_options": {
    "algorithm": "auto",
    "color_distortion_limit": 3,
    "spatial_distortion_limit": 5
  },
  "text_watermarking_enabled": true,
  "text_watermarking_options": {
    "rendering_intent": "screen"
  },
  "visible_watermarking_enabled": true,
  "visible_watermarking_options": {
    "text": "Personalized for {{recipient}}",
    "position": "footnote",
    "color": "000000",
    "opacity": 1
  }
}

-----FormBoundary7MA4YWxkTrZu0gW--j
```

They similarly return marked content as a multipart form returning two pieces of information. The marked content (which is to be delivered to the recipient) and the mark schema. The mark schema is a “recipe” for regenerating a particular marked copy with version information and checksums etc. This data is persisted by EchoMark's API server but it is useful for the developer to log this data as well. It acts as a sort of receipt for marking, and in the event of data loss, can be used to investigate.

```
Content-Type: multipart/form-data; boundary=----FormBoundary7MA4YWxkTrZu0gW
-----FormBoundary7MA4YWxkTrZu0gW
```

```

Content-Disposition: form-data; name="marked_copy"
Content-Type: image/jpeg

[Binary data of the JPEG file]
-----FormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="marking_schema"
Content-Type: application/json

{
  "mark_schema": {
    "mark_options": {
      "recipient": "employeeid_dwwhijtoarkbmkkz",
      "recipient_name": "Alice Bar",
      "image_watermarking_enabled": true,
      "image_watermarking_options": {
        "algorithm": "auto",
        "color_distortion_limit": 3,
        "spatial_distortion_limit": 5
      },
      "text_watermarking_enabled": true,
      "text_watermarking_options": {
        "rendering_intent": "screen"
      },
      "visible_watermarking_enabled": true,
      "visible_watermarking_options": {
        "text": "Personalized for {{recipient}}",
        "position": "footnote",
        "color": "000000",
        "opacity": 1
      }
    },
    "content_type": "image/jpeg",
    "sha256":
      "b94d27b99a1139c96512636b670770f40d25090598f61a113072927934519187",
    "original_seed_checksum":
      "b94d27b99a1139c96512636b670770f40d25090598f61a113072927934519187"
  },
  "version": 2,
  "version_details": "{\"ImageEngineVersion\": \"a927934519187\",
    \"PDFEngineVersion\": \"b94d27b99a1139\"}"
}

-----FormBoundary7MA4YWxkTrZu0gW--j

```

The developer can anticipate the following responses from the API call.

Status	Meaning	Description
200	OK	Successful response with watermarked data
400	Bad Request	Bad request - invalid input where one or more of the marking options is incorrectly specified.

JSON Mode

By default, your container will accept multipart form requests. But, you can use json requests if pass in a URL argument `?json=True` to any of the endpoints below. In this mode, your files will be encoded as a base64 string instead of multipart forms.

```
Content-Type: application/json
{ "original":
  "iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAQAAAC1HAWCAAAAC01EQVR42mNk+A8AAQUBAScY42YAAAAASUVORK5CYII
=",
  "content_type": "image/png",
  "mark_options":
    { "recipient": "employeeid_dwwhijtoarkbmkz",
      "recipient_name": "Alice Bar",
      "image_watermarking_enabled": true,
      "image_watermarking_options": {
        "algorithm": "auto",
        "color_distortion_limit": 3,
        "spatial_distortion_limit": 5
      },
      "text_watermarking_enabled": true,
      "text_watermarking_options": {
        "rendering_intent": "screen"
      },
      "visible_watermarking_enabled": true,
      "visible_watermarking_options": {
        "text": "Personalized for {{recipient}}",
        "position": "footnote",
        "color": "000000",
        "opacity": 1 }
    }
}
```

Similarly, responses to the above requests will be returned as JSON as well. Note that the names of the parts are consistent the only difference is that the file parts are included in the json structure as base64 encoded strings.

```
Content-Type: application/json
{
  "marked_copy":
  "iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAQAAAC1HAWCAAAAC01EQVR42mNk+A8AAQUBAScY42YAAAAASUVORK5CYII
=",
  "mark_schema": {
    "mark_options": {
      "recipient": "employeeid_dwwhijtoarkbmkz",
      "recipient_name": "Alice Bar",
      "image_watermarking_enabled": true,
      "image_watermarking_options": {
        "algorithm": "auto",
        "color_distortion_limit": 3,
        "spatial_distortion_limit": 5
      },
    },
  },
}
```

```

    "text_watermarking_enabled": true,
    "text_watermarking_options": {
      "rendering_intent": "screen"
    },
    "visible_watermarking_enabled": true,
    "visible_watermarking_options": {
      "text": "Personalized for {{recipient}}",
      "position": "footnote",
      "color": "000000",
      "opacity": 1
    }
  },
  "content_type": "image/jpeg",
  "sha256": "b94d27b99a1139c96512636b670770f40d25090598f61a113072927934519187",
  "original_seed_checksum":
    "b94d27b99a1139c96512636b670770f40d25090598f61a113072927934519187"
  },
  "version": 2,
  "version_details": "{ \"ImageEngineVersion\": \"a927934519187\", \"PDFEngineVersion\": \"b94d27b99a1139\" }"
}

```

[/document/mark/pdf](#)

A POST request to this API endpoint with PDF file (application/pdf) will return a marked PDF. PDFs are self-contained documents that contain all of the images and design content in a single file container. EchoMark will mark all content in the PDF including embedded content. For example, if the PDF contains an embedded image, EchoMark will mark the image based on the developer's image watermarking settings. All of this content is considered a single "original" for EchoMark's purposes.

[/document/mark/html](#)

A POST request to this API endpoint with a HTML document will mark the text body of the document with EchoMark, i.e., adjust the positioning of any static HTML text in the DOM. The endpoint also supports marking HTML fragments. This endpoint will not mark images that happen to be on the HTML document in tags or <canvas> tags. Often, these elements refer to external URLs or are populated asynchronously with Javascript. The returning content will be a marked html document. There are a few tips to ensure design parity with the original.

1. Explicitly define preferred fonts for styles and do not rely on system defaults. EchoMark will override problematic system default fonts that are known to cause issues.
2. Avoid absolute positioning and sizing if possible. EchoMark's spacing may cause minor overflows that do not always interact nicely with absolute sizes.

3. Avoid styles that change word/character spacing, e.g., letter-spacing, word-spacing, text-align: justify, etc.

/document/mark/image

A POST request to this API endpoint with an image file (image/*) will return a marked image. If the image contains any layered information, EchoMark will flatten all layers and mark the flattened file. The return type is of the same format as the input. For example, if you post an image with content type image/png the return type is image/png.

Marking Static Workflow

Now, we diagram an example workflow for the static marking API for developers.

1. Developer has a document that needs to be delivered to a recipient.
2. The developer's program calls the appropriate /document/mark/* endpoint with the appropriate mark options.
3. The requestor waits for a response if 200 OK, then the extracts the file body from the multipart form response.
 - a. An explicit error code implies permanent failure.
 - b. A closed connection may indicate the server is busy and the requestor can wait and retry.
4. The requestor delivers the marked file to the end recipient.
5. The developer needs to store the original document (if not already archived) and the marking schema returned by the multipart form response.

Investigations

Investigations resolve a leaked artifact to a particular marked copy and thus a recipient. Here is the intended workflow for the investigation API (called "detect" as a verb). Your organization identifies a potential leak of data. You gather a set of "artifacts" that are leaked. These are supposed copies, renderings, or photographs of marked content. For each artifact, you should identify an original document (or a set of possible originals) from where it could be derived. Additionally, helpful hints such as page numbers can be optionally provided. Investigation involves passing an artifact, a plausible set of originals, and a set of hints to a detect endpoint.

After running a leak investigation, you'll be presented with a leak report. The leak report gives a high level match result, including details about the results.

Match Confidence

Each result comes with an associated chance of error. The chance of error measures EchoMark's confidence in the accuracy of the results. It uses a combination of statistical analysis and extensive offline simulation to ensure trust in the results.

The chance of error is a measure of statistical significance called a p-value, which is a statistical tool to evaluate whether differences between noisy measurements are attributable to random chance. EchoMark measures the spatial correlation of marks in a leaked artifact against the marks we know we introduced into the email. We use a statistical test called the chi-squared test to determine whether the difference in correlation between the top-ranked and second-ranked copies is statistically significant. The confidence score is a defense against distortions in the artifact, i.e., whether the marks that we were unable to detect could change the results of the investigation.

The API will yield a "Possible Match" when an artifact is uploaded for the leak investigation and our detection system finds that it matches a marked copy when the chance of error is low. Sometimes our computer vision detection system is not confident in determining a clear match and will yield a "No Match".

The meaning of a result

It is important to note that while an investigation result can be used to confidently determine that the likely source of the leaked content was from the copy originally sent to the recipient indicated in the results, EchoMark cannot determine the ultimate path it took to getting leaked. It's possible that the leak was not the fault of that recipient holder. Some non-exhaustive possible explanations include (in non-ranked order):

- The recipient leaked the content directly
- Someone else who had permissible or compromised access to the account owner's computer(s) or account leaked the content.
- A copy of the marked content was shared with someone else who ultimately leaked the content.

A full investigation should be run including the gathering of additional forensics information to help validate or rule out various possible explanations for your result.

For Marked Documents

/detect/ endpoint

Detect requests are also passed as multipart forms. It expects a single artifact file named "artifact". It can take any amount of originals. The API server uses the glob "original*" to determine whether a file is an original. Each original is hashed and checked against our internal content store to determine if it has been marked before. Detect will ignore any original that hasn't been marked by our system.

```
Content-Type: multipart/form-data; boundary=----FormBoundary7MA4YWxkTrZu0gW
-----FormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="artifact"
Content-Type: image/png

[Binary content of png image]
-----FormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="original_1"
Content-Type: application/pdf

[Binary content of the pdf document]
-----FormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="original_2"
Content-Type: application/pdf

[Binary content of the pdf document]
-----FormBoundary7MA4YWxkTrZu0gW-j
Content-Disposition: form-data; name="detect_options"
Content-Type: application/json
{
  "investigation_hints": {
    "page_idx": 0,
    "orientation_idx": 1
  },
  "background": "True"
}
-----FormBoundary7MA4YWxkTrZu0gW--j
```

Detect optionally takes in two options: investigation hints, background. Investigation hints provide hints to the investigation engine to speed up the analysis time.

Name	Type	Description
page_idx	integer	Provide a page index (0 indexed) as a hint to the investigation engine. This lets us know which page you think leaked.

orientation_idx	integer	An orientation hint that describes any rotations or reflections needed for detection in the exif convention 1 Top-left (Horizontal/Normal), 2 Top-right (Mirror Horizontal) 3 Bottom-right (Rotate 180 degrees) 4 Bottom-left (Mirror Vertical) 5 Left-top (Mirror Horizontal and Rotate 270 CW) 6 Right-top (Rotate 90 CW) 7 Right-bottom 8 Left-bottom (Rotate 270 CW)
group_originals	boolean	When multiple originals are used run detect against all or run detect against the best match.

If page_idx is provided, it is assumed that all originals provided are page-aligned. If not, leave this blank. Background determines whether the detect task runs synchronously or asynchronously. More on this a bit later.

background	boolean	Whether or not to execute the investigation task in the background. When false the result is synchronously returned to the requestor, when true the request returns synchronously with an analysis_id to query at a later time.
------------	---------	---

Detect returns a [response](#) that looks like this.

Name	Type	Description
scores	ScoredCopy	A json of scores associated with labels
confidence	string	A possible_match result indicates that there is a statistical winner, a no_match result indicates a lack of confidence from the EchoMark algorithm.

Scored copies are tuples of names and scores. The confidence score measures whether the highest ranked (highest absolute score) is a statistical winner or not.

Name	Type	Description
label	string	A label for a copy, e.g., a filename + recipient name, that lets you inspect the outputs of an investigation

score	number	A score from 0 to 1 describing the degree to which the copy matches the artifact.
-------	--------	---

To make this clearer, here is an example /detect payload.

```
{
  "scores": [
    {
      "label": "secure-Alice-Bar.pdf",
      "score": 0.95
    },
    {
      "label": "secure-Bob-Car.pdf",
      "score": 0.65
    }
  ],
  "confidence": "possible_match"
}
```

For Background Detect

Detect tasks can take a substantial amount of time. The background option gives users the option to trigger a detect asynchronously. When false the result is synchronously returned to the requestor, when true the request returns synchronously with an analysis_id to query at a later time. Synchronous detect is useful for debugging or assessing that the system is working correctly. When background detect is set to true, detect returns almost immediately with an analysis id:

```
{
  "analysis_id" : "b94d27b99a1139c96512636b6"
}
```

This analysis id can be passed into another API endpoint to query the report.

JSON Mode

By default, your container will accept multipart form requests. But, you can use json requests if pass in a URL argument ?json=True to the detect endpoint. Detect requests can similarly be sent in JSON mode. The structure is slightly different than that of above with the multipart forms.

Here is the basic schema:

```
Content-Type: application/json
{
  "artifact": <base64 encoded string>
  "content_type": <the content type for the artifact above>
  "investigation_hints": <investigation hints following the structure above>
  "originals": [ <a list of original structs>
```

```

        {
            "original": <base64 encoded string of original document>
            "marked_copies": <an optional list of marking schemas if storage mode is
set to none>
        }
    ]
    "background": <whether to execute in background mode>
}

```

Note: This structure is exactly the same as EchoMark's V1 JSON structure with the only change being additional options of background execution and orientation correction.

Note: Algorithm versioning and background execution with state is not supported in JSON mode.

/reportjson/<analysis id>

The developer can run a get request against a particular analysis id, for example, "GET /reportjson/b94d27b99a1139c96512636b6". The end point will return responses in the following schema.

Status	Meaning	Description
200	OK	Report retrieved successfully
403	Forbidden	Report not found or not complete
500	Internal Error	Investigation resulted in an error.

The user can poll the endpoint until the status returns complete. The result of the report is in the same structure as one expects with synchronous detect.

Example Detect Request:

As described above, to initiate a detect request, you first need to have a few things.

1. An artifact (e.g. an image or PDF) which represents the leaked (marked) copy.
2. A list of all originals from which the artifact could have originated.
3. The marking schema associated with all markings ever created from those originals.

We provide an example detect flow in Python below. First, we provide an example of marking a few documents, in order to clearly illuminate how the marking schemas from step 3 are used in the detect flow.

