UK export Licence: OGE2020/000408

zTeros Light Setup and Developers Guide

zTeros light with MongoDB in one instance

15-AUG-20

zTeros Light

UK export Licence: OGE2020/000408

Contents

UK export Licence: OGE2020/000408

zTeros Light Product

The product discussed in this use case application note is zTeros light.

zTeros light provides a simple single instance solution for data at rest confidentiality and security for individual records on a cloud system. zTeros Light is differs in in approach to data at rest, in that instead of using a single key to encrypt a whole volume, zTeros Light encrypts every record or file each with its own key. What this does is add defence in depth to live operation. Crypt controlled file or record access reduces the impact of data aggregation to single records and provides an opportunity for new application to use keys combined with permissions to enforce access control with strong security mechanisms.

zTeros light is designed to be simple to use, and with only 4 commands within the API, zTeros Light is probably the simplest data confidentiality product ever launched in the cloud computing market. Applications which are, or include, Archiving, Data Storage, Data-migration, Database and AWS data storage solutions, can benefit from the zTeros Light encryption service.

Data Format

If you are using the JSON format for your data, we strongly recommend that you encode it for example in base64. This will make sure that you don't't have to handle your JSON embedded in our own.

As zTeros is a REST API it expects to see strings, not binary, within the body therefore Base64 conversion is required for Binary files and objects, which will be approximately 33% larger after encoding.

The quoted zTeros Maximum Data object sizes in the vertical scaling section are after base 64 encoding, i.e. the size zTeros sees.

Setup and configuration

AWS Launch Instructions:

1. Launch the product via 1-click.
2. Wait for 2 minutes after it has launched (There are post first boot scripts that need a little time to run)

   To check the script has finished, log into the server and run the following command

   ```
   systemctl status glassfish
   ```

   If this confirms that glassfish is running then the script has finished.
3. After the script has finished the installation should be ready for use.

Logging in to MongoDB

As zTeros light uses a Username and Password protected MongoDB setup.

An Admin and an Application Account has been created by the first boot script within mongoDB.

The Admin Account is called "UserAdmin" the Application Account is called "encServiceUser"

The Passwords for both accounts are in /opt/MongoDB_user_details.txt and are randomised for each instance generated during first boot scripts. Root access is required to read this file.

Properties File Location:

/opt/payara5/glassfish/domains/domain1/applications/EncryptionService/WEB-INF/classes/EncryptionService.properties

Default Settings:

Setting	Default	Description
Public_key	/var/lib/encryption_service/pub/public.key	Location of the Public Key, used to Verify UIDs.
Private_key	/var/lib/encryption_service/priv/private.key	Location of the Private Key, used to Sign UIDs.
Cypher_type	AES/GCM/NoPadding	The Cypher (cipher) used when encrypted data.
Hash_algorithm	sha-384	The Hashing Algorithm used within zTeros.
Signing_algorithm	SHA1withECDSA	The Signing Algorithm used within zTeros.
MongoDB_host	localhost	The Host for Mongo DB.
MongoDB_port	27017	The Port Mongo is listening on.
MongoDB_user	encServiceUser	The Database User in Mongo we will be using.
MongoDB_pass	{random}	The Password for said Mongo User.
MongoDB_scram_type	sha256	The SCRAM method used to secure the MongoDB logins
MongoDB_auth	enc	The DB in mongo that has the security information needed to login.
MongoDB_db	enc	The DB zTeros will store data in within the Mongo instance
MongoDB_collection	erd_collection	The Collection zTeros will store data in within the MongoDB instance

Changing MongoDB connection details

Currently zTeros only support SCRAM Authentication to MongoDB. Instructions on how to set this up are available through Mongos Documentation here.
The user needs to have the readWrite role for the DB containing the records to work as we will be reading and writing to this DB.

Scaling zTeros

Vertical Scaling

zTeros can be Vertically scaled by increasing the instance size. Larger instances allow zTeros to handle bigger payloads, or more requests simultaneously.

For new instances this is automatically set to a sensible level, for example 4G instance is set up as Xmx 3072M.

If you change the size of an instance, you will need to manually update the JVM option for -Xmx in /opt/payara5/glassfish/domains/domain1/config/domain.xml.

Instance Name	Instance Total Memory (GiB)	zTeros JVM Memory Allocation (GiB)	Estimated Sustained Message size (constant simultaneous requests)	Estimated Max Message size
T2.micro	1	0.5	20KB	2MB
T2.small	2	1	5MB	50MB
T2.medium	4	3	20MB	150MB
T2.large	8	7	50MB	350MB
T2.xlarge	16	15	100MB	750MB

Simultaneous connections

Please note that Payara (Glassfish) also has a setting on the domain for the "http-thread-pool" and while the default for domain1. On these servers we have set this limit to 20, however if you require more threads running at the same time, you can increase this. We would recommend however to consider doing some Horizontal Scaling if you do need to handle a significant amount of traffic at the same time.

Horizontal Scaling

To scale zTeros Horizontally we recommend that you create a separate Mongo instance for all instances to connect to. Whether you have each instance of zTeros has its own User or there is a single user for all is up to you as long as all users have the same permission for the MongoDB_db setting within the properties file.

All zTeros instances will need to have the same Public and Private Keys as they will be looking at the UIDs. Also, the Cypher_type, Hash_algorithm, and Signing_algorithm will need to match on all instances.

Finally, a Load Balancer would need to be applied for example the Elastic Load Balancing Service from AWS.

Adding a Certificate for HTTPS

We strongly recommend that all connections are HTTPS, unfortunately you have to set up your certificates.

We are running Payara 5 Community Edition, to set up your certificates the Payara 5 documentation is located here .

Methods

Summary

Command	Structure	Description
POST	http(s)://your-url/EncryptionService/records/ Parameters: None Body: None	Creates a new record, returns the RecordUID of the new (public) Example response record {"RecordUID":"12345678abcde", "Completed": "True"}

GET	http(s)://your-url/EncryptionService/records/{RecordUID}
	
	Parameters: None
	Body: None
		Retrieves record {UID}
		
		Example response
		{ "RecordUID": "12345678abcde", "Data": "My Data", "Completed": "True"}

PUT	http(s)://your-url/EncryptionService/records/{RecordUID}
	
	Parameters: None
	Body: New Data in your preferred format.
		Updates record {UID}
		
		Example response
		{ "RecordUID": "12345678abcde", "Completed": "True"}

DELETE	http(s)://your-url/EncryptionService/records/{RecordUID}
	
	Parameters: None
	Body: None
		Deletes record {UID}
		
		Example response
		{"RecordUID": "12345678abcde ", "Completed": "True"}

Example code

Example code for all commands, in Python3, is available on github here:
https://github.com/gbr14/zTerosLightConnectionExample

Create a New Record (POST http(s)://your-url/records/)

Sending a POST request to this URL will create a new blank record in the backend for you to update. Note anything in the Body is ignored in the current version of zTeros and as such you will need to send a PUT request as described further down in order to populate this record.

Returns:
A JSON response of: {"RecordUID":"12345678abcde", "Completed": "True"}

Possible errors:
If the properties file is missing or mis-configured then an Internal Server Error to occur.
If Java is extremely low on memory, we have configured another Internal Server Error to occur, in these instances we recommend applications to try again a few minutes later, if this occurs frequently a larger server is recommended.

Retrieve a record (GET http(s)://your-url/records/{RecordUID})

Sending a GET request to this URL (with a valid and current UID) will retrieve and decrypt a record in the backend for.

Returns:
A JSON response of: {"RecordUID": "12345678abcde", "Data": "My Data", "Completed": "True"}

Possible errors:
If the properties file is missing or mis-configured then an Internal Server Error to occur.

If Java is extremely low on memory, we have configured another Internal Server Error to occur, in these instances we recommend applications to try again a few minutes later, if this occurs frequently a larger server is recommended.

Other Internal Server errors can occur if the server is misconfigured, e.g. the signing keys being in the wrong locations / corrupted. More helpful error messages will appear in the logs. If a UID not signed by the configured certificate is used, then a Not Found (404) error will be returned and it will be logged into the error log.

If a UID that passes the signature check but is not present, most likely a now deleted UID, is requested then a Not Found (404) error will be returned.

Update a Record (PUT http(s)://your-url/records/{RecordUID})

Sending a PUT request to this URL (with a valid and current UID) will update a record in the backend using the of the request as they payload.

Returns:
A JSON response of: {"RecordUID":"12345678abcde", "Completed": "True"}

Possible errors:
If the properties file is missing or mis-configured then an Internal Server Error to occur.
If Java is extremely low on memory, we have configured another Internal Server Error to occur, in these instances we recommend applications to try again a few minutes later, if this occurs frequently a larger server is recommended.

Other Internal Server errors can occur if the server is misconfigured, e.g. the signing keys being in the wrong locations / corrupted. More helpful error messages will appear in the logs. If a UID not signed by the configured certificate is used, then a Not Found (404) error will be returned and it will be logged into the error log.

If a UID that passes the signature check but is not present, most likely a now deleted UID, is requested then a Not Found (404) error will be returned.

Delete a Record (DELETE http(s)://your-url/records/{RecordUID})

Sending a DELETE request to this URL will delete the record in the backend.

Returns:
A JSON response of: {"RecordUID":"12345678abcde", "Completed": "True"}

Possible errors:
If the properties file is missing or mis-configured then an Internal Server Error to occur.
If Java is extremely low on memory, we have configured another Internal Server Error to occur, in these instances we recommend applications to try again a few minutes later, if this occurs frequently a larger server is recommended.

Other Internal Server errors can occur if the server is misconfigured, e.g. the signing keys being in the wrong locations / corrupted. More helpful error messages will appear in the logs. If a UID not signed by the configured certificate is used, then a Not Found (404) error will be returned and it will be logged into the error log.

If a UID that passes the signature check but is not present, most likely a now deleted UID, is requested then a Not Found (404) error will be returned.

An unauthorised error will occur if the record is being used by an zTeros Basic or zTeros Professional instance to store a record with Privileges associated to it.