Cloud Inquisitor Documentation

Release 3.0.0

Riot Games Security

Apr 16, 2020

Contents

This directory has several resources that will help you implement Cloud Inquisitor and contribute to the project.

CHAPTER 1

Getting started

1.1 Quick Start Guide

This tutorial will walk you through installing and configuring `Cloud Inquisitor` ("Cinq"). The tool currently runs on *Amazon Web Services* (AWS) but it has been designed to be platform-independent.

It's highly recommended you first use the Quickstart to build Cinq. However if you want to explore additional options please see additional options.

1.1.1 System Requirements

- Ubuntu 16.04 or higher
- Python 3.5 or higher
- git 2.7 or higher
- make 4.1 or higher
- Be able to `sudo`
- Internet connection

1.1.2 Install Cinq

WARNING: Please using a dedicated system (e.g. VM) as setting up Cinq will result in system changes.

Step 1: Download setup files

You can get everything via cloning Cinq's main repo

```
git clone https://github.com/RiotGames/cloud-inquisitor.git
```

Step 2: Setup necessary parameters

This step is optional if you'd like to set up local development environment. However if you'd like to setup a production server, you might want to set the following environment variables to suit your needs:

Variable	Description
APP_AWS_API_ACCESS_KEY	AWS Access Key ID that will be used by Cinq
APP_AWS_API_SECRET_KEY	AWS Secret Access Key that will be used by Cinq
APP_DB_URI	URI of the Database. e.g. `mysql://cinq:secretpass@127.0.0.1:3306/cinq` This option is mandatory if you want to set up a Production Cinq server
APP_WORKER_PROCS	How many concurrent workers Cinq will run
PATH_CINQ	Directory you want to have Cinq installed
USE_HTTPS	Enable HTTPS or not

Step 3: Setup Cinq, Part I

Depending on your need, please choose the option which suits you best.

Note: Please avoid running `make` under root account directly.

Option A: Local development instance

Go to the root folder of the Cinq repo you just cloned (there should be file named **LICENSE**) then use the following command to setup Cinq

```
sudo -E make setup_localdev
```

Option B: Production Cinq server

1. Make sure you can connect to your database
2. Go to the root folder of the Cinq repo you just cloned (there should be file named **LICENSE**) then use the following command to setup Cinq

```
sudo -E make setup_server_install
sudo -E make init_service_mysql
sudo -E make init_cinq_db
```

Step 4: Setup Cinq, Part II

You should have a file named `cloud-inquisitor` under `{PATH_CINQ}/cinq-venv/bin` which is the main executable you will use to launch Cinq. (If you didn't modify the PATH_CINQ and PATH_VENV environment variables, the default path will be `/opt/cinq/cinq-venv/bin/cloud-inquisitor`)

In this section we will use the default path. If you installed Cinq to a different directory, please modify the commands accordingly.

1. Run the following command

```
/opt/cinq/cinq-venv/bin/cloud-inquisitor runserver
```

2. You will see a lot of output as the result of the initialization. By the end of the output you should see something like below

```
... Cinq output ...
[09:44:02] cinq_auth_local Created admin account for local authentication,␣
↪username: admin, password: LcaJxseObTHRgimWuLywb+ICeoggNpbo
... Cinq output ...
```

3. Take note of the username and the password displayed. It will only display ONCE and cannot be recovered if you lost it.
4. Open your web browser, enter the host name or IP of the server you used to setup Cinq (For local dev instance it should be `127.0.0.1`, use the username and password you just got to login.
5. You should be able to see the web console of Cinq.

Next Steps

Read *Manual* and add your accounts into Cinq so it can start working for you!

1.2 Manual

1.2.1 Project Overview

Backend

This project provides two of the three pieces needed for the Cloud Inquisitor system, namely the API backend service and the scheduler process responsible for fetching and auditing accounts. The code is built to be completely modular using `pkg_resource` entry points for loading modules as needed. This allows you to easily build third-party modules without updating the original codebase.

API Server

The API server provides a RESTful interface for the frontend web client.

Authentication

The backend service uses a JWT token based form of authentication, requiring the client to send an Authorization HTTP header with each request. Currently the only supported method of federated authentication is the OneLogin based SAML workflow.

There is also the option to disable the SAML based authentication in which case no authentication is required and all users of the system will have administrative privileges. This mode should only be used for local development, however for testing SAML based authentications we have a OneLogin application configured that will redirect to http://localhost based URL's and is the preferred method for local development to ensure proper testing of the SAML code.

More information can be found at:

- SAML
- Local

Auditors

Auditors are plugins which will alert and potentially take action based on data collected.

Cloudtrail

The CloudTrail auditor will ensure that CloudTrail has been enabled for all accounts configured in the system. The system will automatically create an S3 bucket and SNS topics for log delivery notifications. However, you must ensure that the proper access has been granted to the accounts attempting to log to a remote S3 bucket. SNS subscriptions will need to be confirmed through an external tool such as the CloudTrail app.

More information such as configuration options here.

Domain Hijacking

The domain hijacking auditor will attempt to identify misconfigured DNS entries that would potentially result in third parties being able to take over legitimate DNS names and serve malicious content from a real location.

This auditor will fetch information from AWS Route53, CloudFlare, and our internal F5 based DNS servers and validate the records found against our known owned S3 buckets, Elastic BeanStalks, and CloudFront CDN distributions.

More information such as configuration options here.

IAM

The IAM roles and policy auditor will audit, and if enabled, will manage you AWS policies stored in Github.

More information such as configuration options here.

Required Tags

Cloud Inquisitor audits EC2 instances and S3 Buckets for **tagging compliance** and shutdowns or terminates resources if they are not brought into compliance after a pre-defined amount of time.

More information such as configuration options here.

Note: This is currently being extended to include all taggable AWS objects.

Default Schedule for Resources that can be Shutdown

Age	Action
0 days	Alert the AWS account owner via email.
21 days	Alert the AWS account owner, warning that shutdown of instance(s) will happen in one week
27 days	Alert the AWS account owner, warning shutdown of instance(s) will happen in one day
28 days	Shutdown instance(s) and notify AWS account owner
112 days	Terminate the instance and notify AWS account owner

Default Schedule for Resources that can only be terminated (S3, ECS, RDS...)

Age	Action
0 days	Alert the AWS account owner via email.
7 days	Alert the AWS account owner, warning termination of resource(s) will happen in two weeks
14 days	Alert the AWS account owner, warning shutdown of resources(s) will happen in one week
20 days	One day prior to removal, a final notice will be sent to the AWS account owner
21 days	Delete* the resource and notify AWS account owner

* For some AWS resources that may take a long time to delete (such as S3 buckets with terabytes of data) a lifecycle policy will be applied to delete the data in the bucket prior to actually deleting the bucket.

S3 Bucket Auditor

S3 Buckets have a few quirks when compared to EC2 instances that must be handled differently. * They cannot be shutdown, only deleted * They cannot be deleted if any objects or versions of objects exist in the bucket * API Calls to delete objects or versions in the bucket are blocking client-side, which makes deleting a large number of objects from a bucket (100GB+) unreliable

Because of this, we have decided to delete contents of a bucket by using lifecycle policies. The steps the auditor takes when deleting buckets are:

1. Check to see if the bucket has any objects/versions. If it's empty, delete the bucket .
2. If the bucket is not empty, iterate through the lifecycle policies to see if our policy is applied.
3. If the lifecycle policy does not exist, apply the lifecycle policy to delete data .
4. If a bucket policy to prevent s3:PutObject and s3:GetObject does not exist on the bucket, apply that policy.
5. If a lifecycle policy to delete version markers does not exist, apply the policy to delete version markers.

This covers a few different edge cases, most notably it allows the auditor to continuously run against the same bucket with re-applying the same policies, even if the bucket contains terabytes of data. Applying bucket policies to prevent s3:PutObject and s3:GetObject prevents objects from being added to the bucket after the lifecycle policy has been applied, which would lead to the bucket never being deleted.

The default expiration time of objects for the lifecycle policy is three days. If this bucket is being used as a static website or part of any critical service, this gives the service owners immediate visibility into the actions that will be soon be taken (bucket deletion) without permanently deleting the content. Although at this point the bucket is non-compliant and should be deleted, being able to reverse a live service issue caused by the tool is more important than immediately and irrecoverably deleting data.

If a bucket is tagged properly after the lifecycle policy has already been applied and the bucket has been marked for deletion, the auditor will not remove the policies on the bucket. The bucket policy and lifecycle policy must be removed manually.

At this point in time, the policy itself is not checked to ensure that it matches the one that we apply. This allows a user to create a policy with a name that matches our policy, and it would prevent their bucket from being deleted. At this time we treat it as an edge case similar to enabling EC2 instance protection, but plan to fix it in the future.

Collectors

Collectors are plugins which only job is to fetch information from the AWS API and update the local database state.

AWS

The base AWS collector queries all regions for every account collecting information for all regions in each AWS account.

A more detailed description is available here.

DNS

The DNS collector gathers and collates all related DNS information, with which the relevant DNS auditors can analyse for potential security issues.

A more detailed description is available here.

Frontend

This project provides the web frontend for the `Cloud Inquisitor` system, and is built using AngularJS and AngularJS Material with a few jQuery based libraries as well.

Building

The code is built using npm and gulp.

To get started building a working frontend, you need to first ensure you have `NodeJS` and `npm` installed and then run the following commands:

```
bash
cd $Cloud-Inquisitor-REPO/frontend
npm install
node_modules/.bin/gulp
```

This will result in production-ready (minified) HTML and Javascript code which can be found in the `dist` folder.

Additional Options

Databases

Cinq is currently designed to run with MySQL Version 5.7.17. We recommend you stick to this version.

If you do not wish to use a local MySQL DB that the Cinq install gives you, in your variables file, simply set the following in your variables before you run the packer build to disable the install and setup of the local DB and point to the database you'd like to use

```
"app_setup_local_db":      "False"
"app_db_uri":              "mysql://<user>:<pass>@<hostname>:3306/<yourdb>"
```

Once the AMI is created and you've logged in you'll need to initialize the database. In order to do so execute the following commands

```
# source /path/to/pyenv/bin/activate
# export INQUISITOR_SETTINGS=/path/to/cinq-backend/settings/production.py
# cd /path/to/cinq-backend
```

(continues on next page)

(continued from previous page)

```
# cloud-inquisitor db upgrade
# python3 manage.py setup --headless
```

You may receive some warnings but these commands should succeed. Then if you restart supervisor you should be good to go

```
# supervisorctl restart all
```

You can look in /path/to/cinq-backend/logs/ to see if you have any configuration errors.

KMS and UserData

You may not wish to keep database credentials in flat configuration files on the instance. You can KMS encrypt these variables and pass them to the Cinq instance via AWS userdata. In your variables file use the following

```
"app_use_user_data":                    "True",
"app_kms_account_name":                 "aws-account-name",
```

When you launch the AMI packer created, you can encrypt the APP_DB_URI setting

```
$ aws kms encrypt --key-id arn:aws:kms:us-west-2:<account_id>:key/xxxxxxxx-74f8-4c0c-
↪be86-a6173f2eeef9 --plaintext APP_DB_URI="mysql://<user>:<pass>@<hostname>:3306/
↪<yourdb>"
```

It will return a response with a field of CipherTextBlob that you can paste into your UserData field when you launch the AMI.

To verify your Cinq instance is using KMS, your production settings in `/path/to/cinq-backend/settings/production.py` should contain:

```
USE_USER_DATA = True
KMS_ACCOUNT_NAME = '<account_name>'
USER_DATA_URL = 'http://169.254.169.254/latest/user-data'
```

Authentication Systems

Cinq supports built-in authentication system (default), as well as federation authentication with OneLogin IdP via SAML. It's possible that other IdPs can be used but this has not been tested.

Edit your `/path/to/cinq-backend/settings/settings.json` file and provide the required values:

```
# source /path/to/pyvenv/bin/activate
# cd /path/to/cinq-backend
# cloud-inquisitor auth -a OneLoginSAML
  cloud_inquisitor.plugins.commands.auth Disabled Local Authentication
  cloud_inquisitor.plugins.commands.auth Enabled OneLoginSAML
```

Verify that your configuration is correct and the active system

```
# cloud-inquisitor auth -l

cloud_inquisitor.plugins.commands.auth --- List of available auth systems ---
cloud_inquisitor.plugins.commands.auth Local Authentication
```

(continues on next page)

(continued from previous page)

```
cloud_inquisitor.plugins.commands.auth OneLoginSAML (active)
cloud_inquisitor.plugins.commands.auth --- End list of Auth Systems ---
```

To switch back to local Auth simply execute

```
# cloud-inquisitor auth -a "Local Authentication"
```

Additional Customization

In the packer directory, the build.json contains other parameters that you can modify at your discretion.

Packer Settings

- `aws_access_key` - Access Key ID to use. Default: *AWS_ACCESS_KEY_ID* environment variable
- `aws_secret_key` - Secret Key ID to use. Default: *AWS_SECRET_ACCESS_KEY* environment variable
- `ec2_vpc_id` - ID of the VPC to launch the build instance into or default VPC if left blank. Default: *vpc-4a254c2f*
- `ec2_subnet_id` - ID of the subnet to launch the build instance into or default subnet if left blank. Default: *subnet-e7307482*
- `ec2_source_ami` - AMI to use as base image. Default: *ami-34d32354*
- `ec2_region` - EC2 Region to build AMI in. Default: *us-west-2*
- `ec2_ssh_username` - Username to SSH as for AMI builds. Default: *ubuntu*
- `ec2_security_groups` - Comma-separated list of EC2 Security Groups to apply to the instance on launch. Default: *sg-0c0aa368,sg-de1db4ba*
- `ec2_instance_profile` - Name of an IAM Instance profile to launch the instance with. Default: *CinqInstanceProfile*

Installer Settings

- `git_branch` - Specify the branch to build Default: *master*
- `tmp_base` - Base folder for temporary files during installation, will be created if missing. Must be writable by the default ssh user. Default: */tmp/packer*
- `install_base` - Base root folder to install to. Default: */opt*
- `pyenv_dir` - Subdirectory for the Python virtualenv: Default : *pyenv*
- `frontend_dir` - Subdirectory of *install_base* for frontend code. Default: *cinq-frontend*
- `backend_dir` - Subdirectory of *install_base* for backend code. Default: *cinq-backend*
- `app_apt_upgrade` - Run *apt-get upgrade* as part of the build process. Default: *True*

Common Settings

- `app_debug` - Run Flask in debug mode. Default: *False*

Frontend Settings

- `app_frontend_api_path` - Absolute path for API location. Default: */api/v1*
- `app_frontend_login_url` - Absolute path for SAML Login redirect URL. Default: */saml/login*

Backend Settings

- `app_db_uri` - **IMPORTANT:** Database connection URI. Example: `mysql://cinq:changeme@localhost:3306/cinq`
- `app_db_setup_local` - This tells the builder to install and configure a local mysql database. Default - null
- `app_db_user` - Mysql username. Default - null
- `app_db_pw` - Mysql password. Default - null
- `app_api_host` - Hostname of the API backend. Default: `127.0.0.1`
- `app_api_port` - Port of the API backend. Default: `5000`
- `app_api_workers` - Number of worker threads for API backend. Default: `10`
- `app_ssl_enabled` - Enable SSL on frontend and backend. Default: `True`
- `app_ssl_cert_data` - Base64 encoded SSL public key data, used if not using self-signed certificates. Default: `None`
- `app_ssl_key_data` - Base64 encoded SSL private key data, used if not using self-signed certificates. Default: `None`
- `app_use_user_data` - Tells Cinq to read variables from encrypted user-data
- `app_kms_account_name` - Provides an account name for kms.
- `app_user_data_url` - URL where user data is access. Default: `http://169.254.169.254/latest/user-data`

FYI

The vast majority of these settings should be left at their default values unless you fell you must change them to get Cinq running.

1.2.2 User Guide

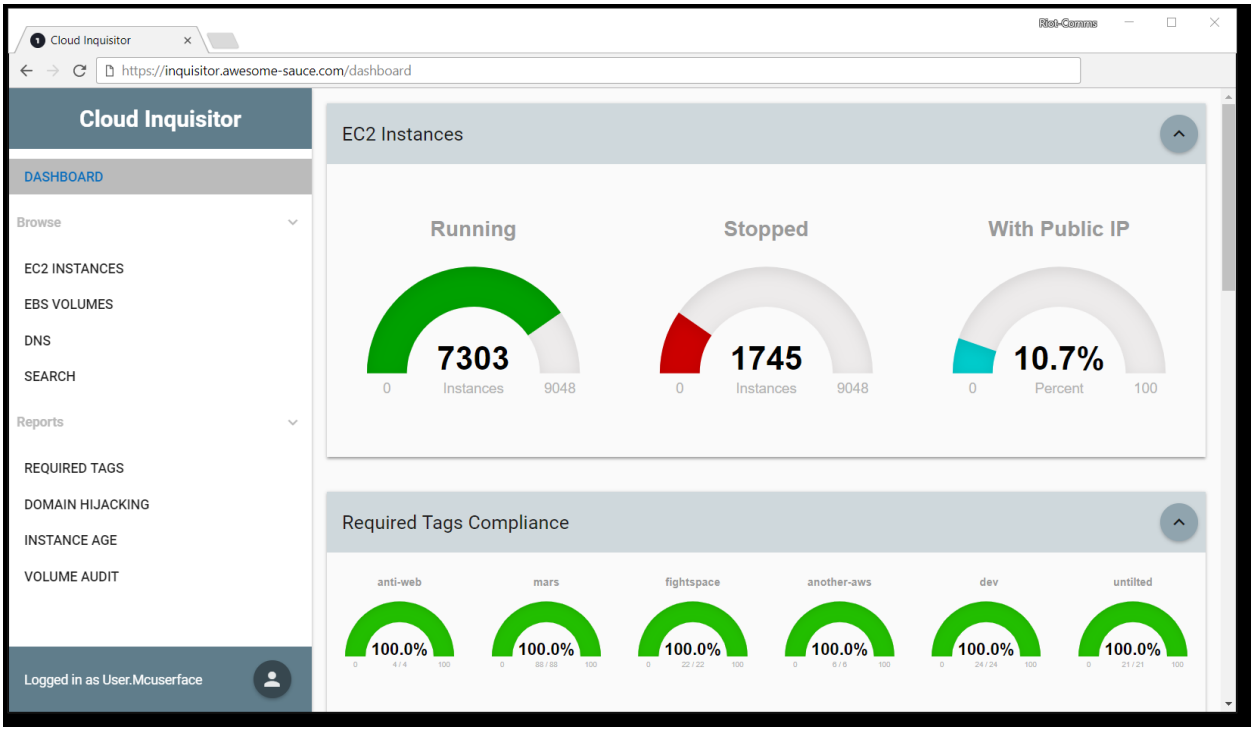
This document is intended to be a user guide to inform on how to use the Cloud Inquisitor UI.

Dashboard

By default, the front-end dasbhoard shows:

- EC2 Instances that are running or stopped and which instances have a public IP.
- Percentage of `required tags` compliance per account.

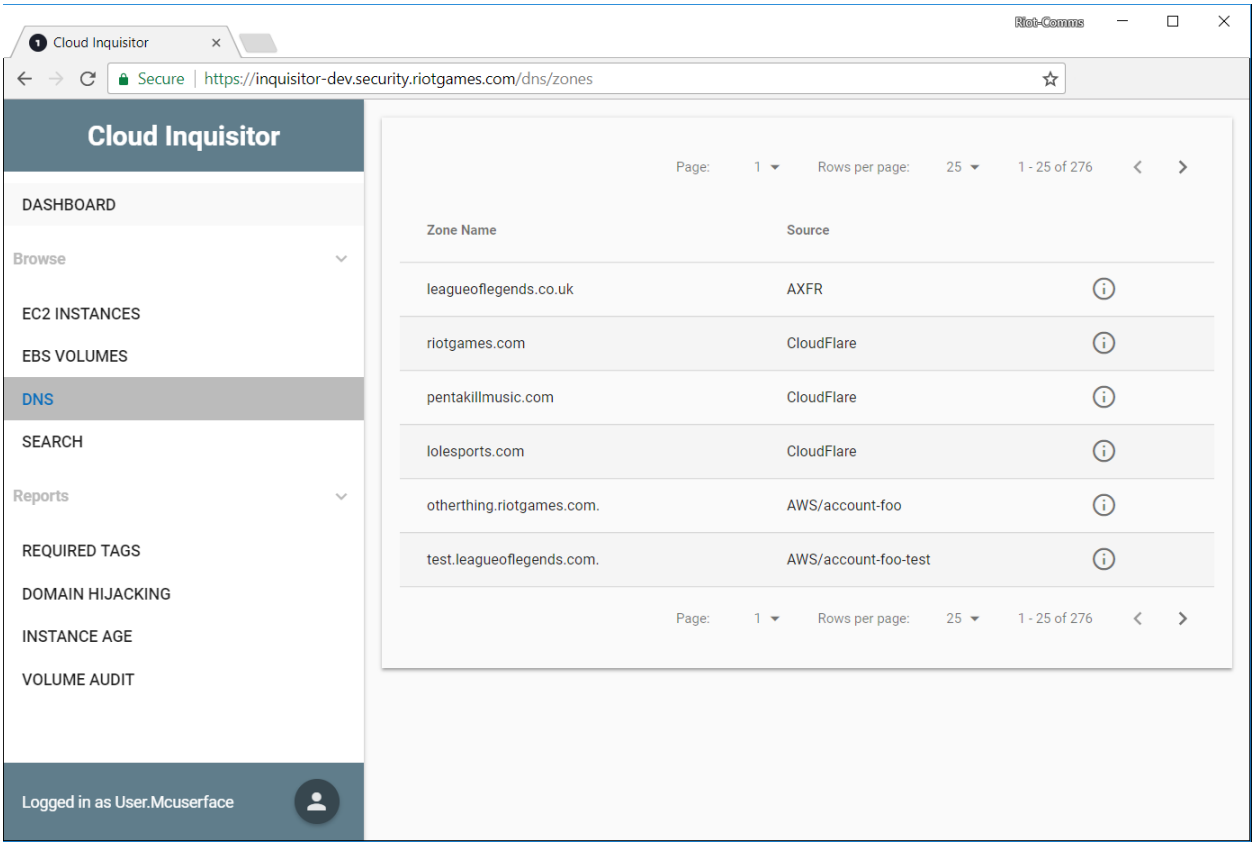
Below is a sample screenshot showing what the dashboard looks like:

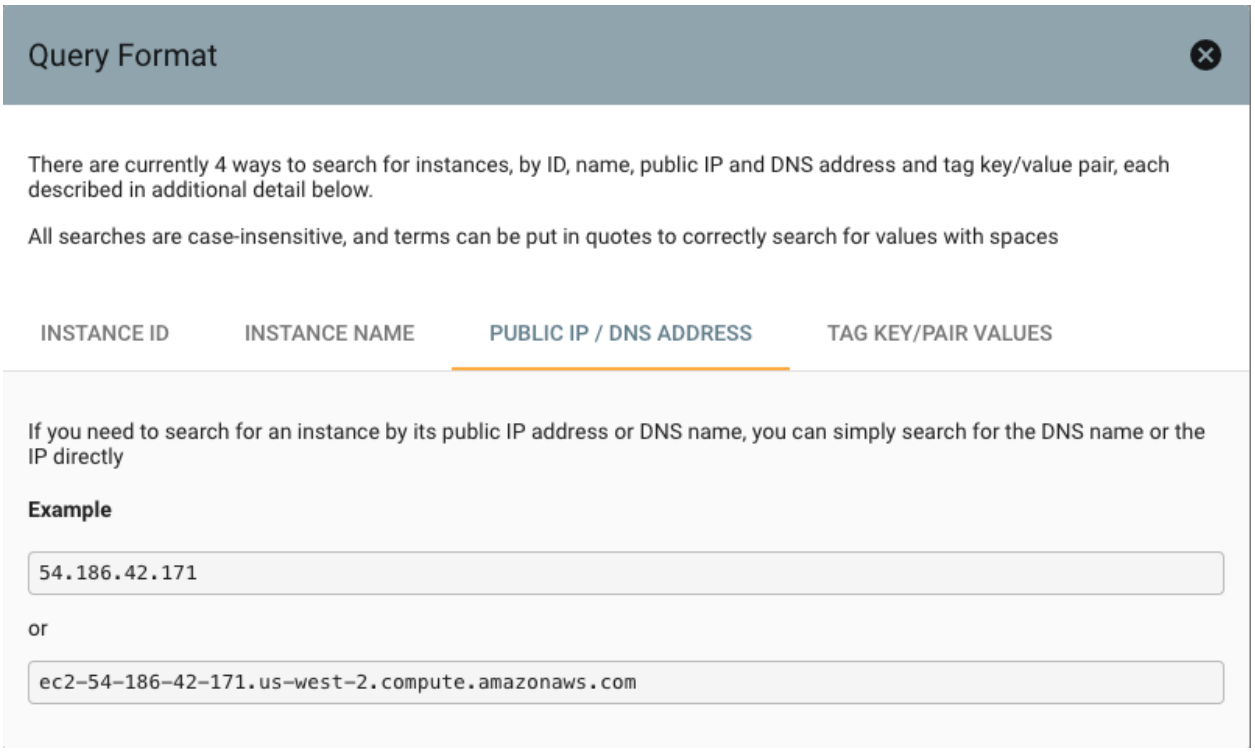


Browse

On the left-hand side of the UI, you are able to directly examine raw data:

- EC2 Instances - shows all the EC2 Instance data that Cloud Inquisitor possess, which should represent all EBS volumes in use in your AWS infrastructure
- EBS Volumes - shows all the EBS Volume data that Cloud Inquisitor possess, which should represent all EBS volumes in use in your AWS infrastructure
- DNS - shows all the dns data that Cloud Inquisitor possess (shown below, first screenshot)
- Search - this gives you the ability to search for instances across the Cloud Inquisitor database. The `search` page has help functionality within the page as shown below (second screenshot)


Cloud Inquisitor
Secure | https://inquisitor-dev.security.riotgames.com/dns/zones
Cloud Inquisitor
DASHBOARD
Browse
EC2 INSTANCES
EBS VOLUMES
DNS
SEARCH
Reports
REQUIRED TAGS
DOMAIN HIJACKING
INSTANCE AGE
VOLUME AUDIT
Logged in as User.Mcuserface
Page: 1 Rows per page: 25 1 - 25 of 276
Zone Name
Source
leagueoflegends.co.uk AXFR
riotgames.com CloudFlare
pentakillmusic.com CloudFlare
lolesports.com CloudFlare
otherthing.riotgames.com. AWS/account-foo
test.leagueoflegends.com. AWS/account-foo-test
Page: 1 Rows per page: 25 1 - 25 of 276


Query Format
There are currently 4 ways to search for instances, by ID, name, public IP and DNS address and tag key/value pair, each described in additional detail below.
All searches are case-insensitive, and terms can be put in quotes to correctly search for values with spaces
INSTANCE ID
INSTANCE NAME
PUBLIC IP / DNS ADDRESS
TAG KEY/PAIR VALUES
If you need to search for an instance by its public IP address or DNS name, you can simply search for the DNS name or the IP directly
Example
54.186.42.171
or
ec2-54-186-42-171.us-west-2.compute.amazonaws.com

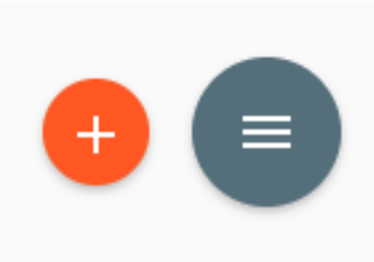
Administration

When logged in as a user with the Admin role, you will see an extra list of sections in the side-menu

- Accounts
- Config
- Users
- Roles
- Emails
- Audit Log
- Logs

Accounts

In this section you can review the current accounts that `Cloud Inquisitor` is auditing and modify accordingly. For example, to add a new account, click the floating menu button in the bottom right hand side of the window and select the "+" as shown below:



Config

In this section you can modify the configuration of both the core platform, as well as all the plugins you have installed. Some plugins may require extensive configuration before you will be able to use them, while others will have usable defaults and not require much configuration.

Below is a list of the configuration options for the core system

Default

Option	Description	Default Value
auth_system	Controls the currently enabled authentication system.	`Local Authentication`
ignored_aws_regions_regexp	Regular expression of AWS region names to NOT include in the list of regions to audit	`(^cn-|GLOBAL|-gov|eu-north-1)`
jwt_key_file_path	Path to the SSL certificate used to sign JWT tokens	`ssl/private.key`
role_name	Name of the AWS Role to assume in remote accounts	`cinq_role`
scheduler	Name of the scheduler system to use	`StandaloneScheduler`

Logging

Option	Description	Default Value
enable_syslog_forwarding	Also send application logs to a syslog server	`Disabled`
log_keep_days	Number of days to keep logs in database	`31`
log_level	Minimum severity of logs to store	`INFO`
remote_syslog_server_addr	Hostname or IP address of syslog server	`127.0.0.1`
remote_syslog_server_port	Port to send syslog data to	`514`

API

Option	Description	Default Value
host	Address for the API to listen on. *Note* this should be kept as localhost / 127.0.0.1 unless the API server is running on a separate machine from nginx	`127.0.0.1`
port	Port to run the API backend on	`5000`
workers	Number of HTTP workers for gunicorn to run	`6`

1.3 Contributing

This directory has several resources that will help you contribute to the project.

1.3.1 Contributing Guidelines

We would love contributions to Cloud Inquisitor - this document will help you get started quickly.

Docs

Within this **docs** directory, you'll find documentation on:

- *Quick Start Guide*
- Issue Log
- Roadmap for Cloud Inquisitor

Submitting changes

- Code should be accompanied by tests and documentation.
- Code should follow the existing style. We try to follow PEP8.
- Please write clear and useful commit messages. Here are three blog posts on how to do it right:
- Writing Git commit messages
- A Note About Git Commit Messages
- On commit messages

- We would prefer one branch per feature or fix; please keep branches small and on topic.
- Send a pull request to the `dev` branch. See the GitHub pull request docs for further information.

Additional resources

- GitHub documentation

1.3.2 Misc Resources

Directory Map

Directory	Description
/backend/	Stores the core backend code
/cinq-venv/	Will be generated during installation. Stores the Python Virtual Environment Cinq uses
/frontend/	Stores the frontend code
/plugins/	Stores the plugins
/resources/	Stores the resources and configurations used by Cinq

1.4 Contributors

- Asbjorn Kjaer (Main Author)
- Marty Chong
- Mark Hillick
- Carl Rutherford
- Gabe Friedmann
- Dong Liu
- Reza Nikoopour
- Zach Pritchard

CHAPTER 2

External resources

- Source Code
- Roadmap
- Issue Tracker
- Join our Slack Chat Room