Cloud Suitability Analyzer (CSA) User Manual

Date	Action	Author
Dec 24, 2025	Retrieved and fixed version 4.1.19	Marc Zottner
Nov 15, 2023	Update with new feature released in 4.1.10	Tanzu Labs
Jul 26, 2023	Update with new features released in 4.0	Tanzu Labs
Feb 10, 2021	Add call graphs to user manual with package descriptions	Tanzu Labs
Jul 30, 2020	Amended rule import instructions	Tanzu Labs
Jul 30, 2020	Prep for open source release	Tanzu Labs
Dec 12, 2018	Converted from readme	Tanzu Labs
Dec 18, 2018	Added scoring/graphics	Tanzu Labs
Dec 20, 2018	Added bucketing and profile	Tanzu Labs
Jan 03, 2018	Added EULA	Tanzu Labs

License

Cloud Suitability Analyzer is licensed under `SPDX-License-Identifier: BSD-2`

Purpose

`csa` is built to automatically scan for potential cloud remediation issues (`TAS`) and cloud accommodation issues (`TKG`) embedded in legacy applications. Currently, rules target Java and .Net, however, any language can be targeted by writing rules that identify patterns for that language or platform.

`csa` is entirely data driven using rules comprised of patterns that are first written in `yaml` and then compiled in the `csa` command-line executable. The rule system is flexible and can scan any type of written text, including source code, configuration files, and xml files. Basically, if the file is human-readable text, a rule can be devised that scans the file.

The matching of patterns and lines of application code require millions of pattern comparisons for each portfolio. To ensure performance of scans, `csa` is built to operate in a highly parallel manner. It is built in the Go language, which produces native-code executables for OSX, Windows, and Linux. `csa` will saturate all the CPUs of its host. Accordingly, `csa` benefits from running on multi-CPU machines, conversely, it suffers if it does not have multiple CPUs. ***We recommend at least a 4-core (8 CPU) machine with 16 gig of RAM.***

The patterns are used to perform global scans of all application files, recursively in the directory specified on the command line. The rules are meant to be curated, and over time will change to adapt to the patterns found on cloud migration and containerization engagements. The intention is to create a single composite score that can be used assess cloud suitability, but also, to provide insights into an applications readiness for containerization.

Binaries/scripts

`csa` binaries run on the following platforms:

Executable	Platform
`csa.exe`	Windows
`csa-l`	Linux
`csa`	OSX

Installation

Download from here:

https://github.com/maarc/csa/releases

There is no real installation process. It is just a matter of deciding on a home directory and copying the files in the `csa` distribution to that directory.

Setting up environment

To effectively use `csa` from the command-line, it will be helpful not to type in the full path every time. So include `csa's` location in your path.

Adding the path on Linux Change to your home directory.

```
cd $HOME
```

Open the `.bashrc` file with a text editor.

Add the following line to the file. Replace the with the location directory of `csa`

```
export PATH=<csa directory>:$PATH
```

Save the file and exit.

Use the source command to force Linux to reload the .bashrc file which normally is read only when you log in each time.

```
source .bashrc
```

Adding the path on OSX Change to your home directory.

```
cd $HOME
```

Open the `.bash_profile` file with a text editor.

Add the following line to the file. Replace the with the location directory of `CSA`

```
export PATH=<csa directory>:$PATH
```

Save the file and exit.

Use the source command to force OSX to reload the `.bash_profile` file which normally is read only when you log in each time.

```
source .bash_profile
```

Adding path on Windows Instructions to change your PATH on Windows 10

File handles If you are experiencing errors such as `Too many open files` or `Unable to open database file` you need to increase you open files /maxfiles ulimit.

If you are attempting to run `csa` on a large directory or set of directories this limit need to be set very high.

MAC OSX Sierra/High Sierra Limited Temporary Fix

Set ulimit to 20000

```
ulimit -n 20000
```

> **Note**: This fix will only live as long as the current shell!

Persistent Fix ***(more flexible. . . lets you set the limit higher. Requires Reboot!)***

1. You have to create a file in your root Library directory. Specifically => `/Library/LaunchDaemons/limit.maxfiles.plis`

   Note: ensure (owner: root:wheel, mode: 0644) see steps below

2. Place the following into the file (set the limits as your desire). The settings below were tested against a portfolio of 36317 files with 7,331,920 lines of code.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
        "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Label</key>
    <string>limit.maxfiles</string>
    <key>ProgramArguments</key>
    <array>
      <string>launchctl</string>
      <string>limit</string>
      <string>maxfiles</string>
      <string>262144</string>
      <string>524288</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>ServiceIPC</key>
    <false/>
  </dict>
</plist>
```

3. Make sure to set the permissions on the file correctly

```
=> sudo chown root:wheel /Library/LaunchDaemons/limit.maxfiles.plist
=> sudo chmod 0644 /Library/LaunchDaemons/limit.maxfiles.plist
```

4. Reboot your machine

Getting help from `csa`

`csa` has several major operating modes with their own associated commands. You can see a list of the command by using `csa help`

Command	Description
bins	Controls the creation of bins, which group similar applications together
rules	Add, edit, delete rules in the field, no need to rebuild executable.
naturalize	Future feature design to use machine learning to find bounded contexts
git	Run git forensics reports
search	Access the indexed search capabilties from the command line
analyze	Default command, scan a directory tree and apply rules
recalculate	Recalculate scores for a prior run
ui	Launch a local web server listening at localhost:3001

If you want help on any of these commands simple type `csa help` and the command name, such as:

`csa help rules`

Generate HTML and CSV reports

`csa` can generate finding reports from command line after each scan as HTML or CSV files (Version 4.0 and higher).

Command	Description
export	List of expected formats divided by commas that will be used to export findings, ex: csv or csv,html
export-dir	Directory path where csa finding exports will be written
export-file-name	Base name of the "export" file, ex: "csa-export". Proper extensions will be appended based on "–export" command formats requested.

```
./csa --export=csv --export-file-name=finding-report --export-dir=[Some Folder Path]
```

This will produce a file "finding-report.csv"

Cloning portfolios

`csa` expects to find a single application per sub-directory, if there are additional application in directory beneath the top directory, they will be considered as one application. This behavior can be controlled using configuration files. See below.

Using configuration files

Configuration files give you full control over how `csa` processes your application portfolio.

The table below describes the settings that are available:

Setting	Description
runName	Specify a run number, `.` lets the number be set by `csa`
applications	A collection of application meta-data
Name	The name of the application, overrides directory name
Path	Directory where your application exists
business-domain	The domain or the department/region of the application
business-value	A number that indicates the value of the app to the business
dir-exclude-regex	A regex that describes directories that should be ignored
include-file-regex	A regex that includes files from processing
exclude-file-regex	A regex that excludes files from processing

Sample file

```
{
    "runName": ".",
    "applications": [
        {
            "Name": "App1",
            "Path": "/Users/user/pvtl/portfoliosmall-shortNames/App1",
            "business-domain": "",
            "business-value": 0,
            "dir-exclude-regex": "^([.].*|target|classes|bin|test|node_modules|eclipse|out)$",
            "include-file-regex": ".*",
            "exclude-file-regex": "^(.*[.](exe|png|tiff|tif|gif|jpg|jpeg|bmp|dmg|mpeg)|[.].*|CSA-config
        },
        {
```

```
        "Name": "App8",
        "Path": "/Users/user/pvtl/portfoliosmall-shortNames/App8",
        "business-domain": "",
        "business-value": 0,
        "dir-exclude-regex": "^([.].*|target|classes|bin|test|node_modules|eclipse|out)$",
        "include-file-regex": ".*",
        "exclude-file-regex": "^(.*[.](exe|png|tiff|tif|gif|jpg|jpeg|bmp|dmg|mpeg)|[.].*|CSA-config
    }
```

Scoring system

Think of the scoring system as a measurement of relative effort to remediate an application to cloud-readiness. We use three loosely applied scales aligned with how often we expect to find a particular pattern in an applications source code.

Occurance	Score Range
Once per application	100-1000
Once per file	10 - 100
Multiple times per file	1 - 10

If the finding is really a positive, such as the discovery of spring boot pattern, then we make the number a negative. Since all scores are subtracted from a perfect score of `10`, a negative score is essentially a positive.

For each application, once we add up the counts multiplied by the score we typically find a very wide range of scores between applications. Some may score a 50, while others may score 30,000. This stems from the fact that scoring is driven by lines of code per file. File size in software follows a `log-normal` distribution. So if we count anything related to file size, we will get a log-normal distribution. It looks like this:

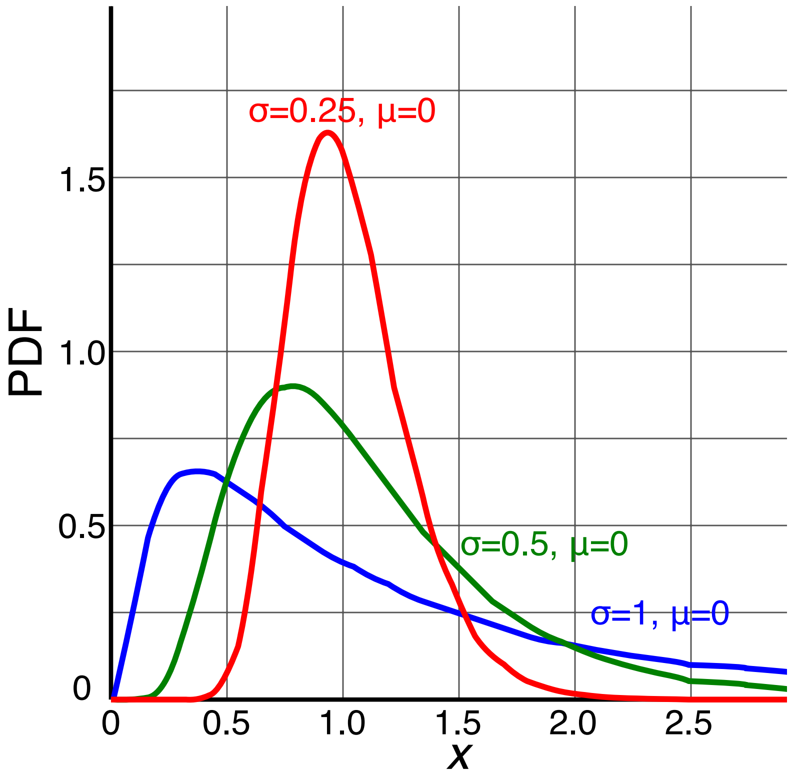


Figure 1: Log-normal distribution

If we want to take the average, the median, or the standard deviation we need a normal distribution. A normal distribution looks like this:

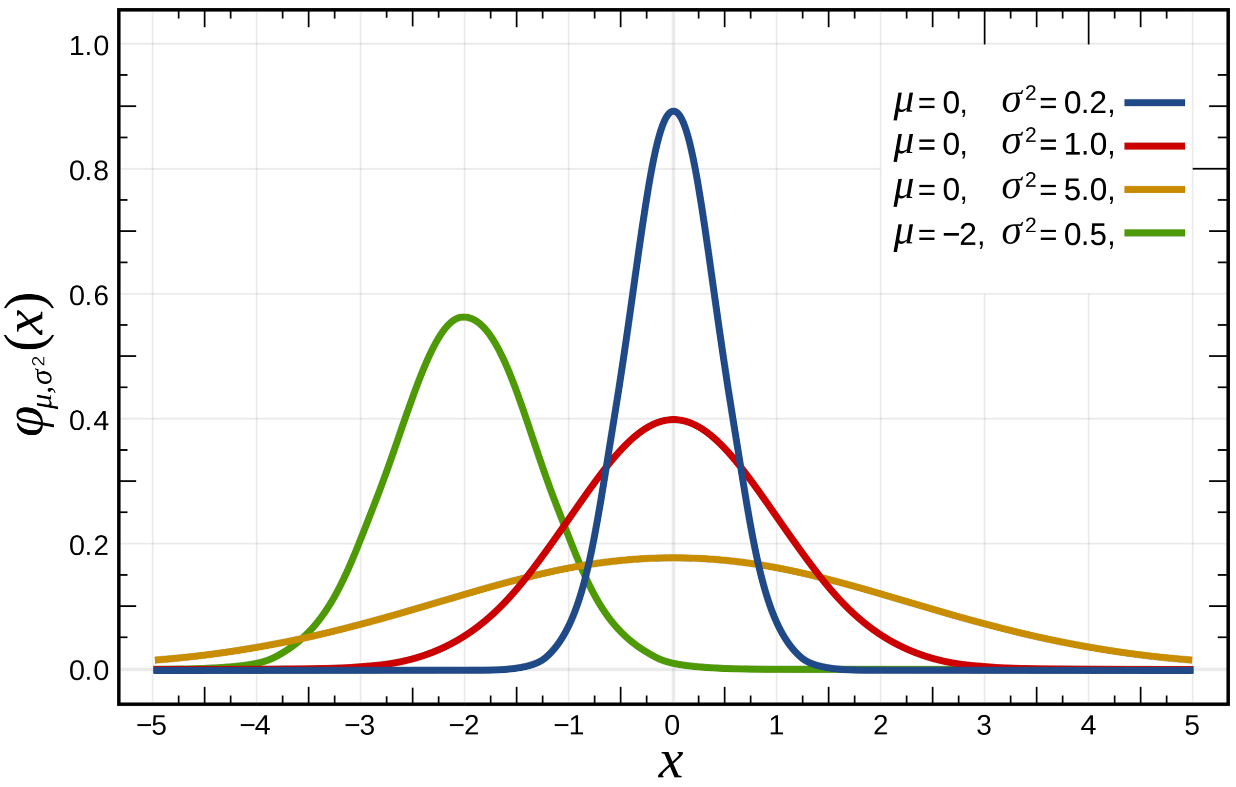


Figure 2: Normal distribution

It's shaped like a bell and is sometimes called a bell-curve. Ever have a teacher that said they would grade you on a curve? This is that curve. Many human phenomena share this distribution. Take a room full of randomly selected people and ask them their height, you'll get a normal curve.

How do we reshape our curve? Since it is `log-normal`, really normal with a skew, we need a way to reshape it. This is very simple, we just take the base 10 logarithm of each number, in this case our total effort score.

Default Model

As discussed earlier, `csa's` scoring system is externalized into a `yaml` files. The model also includes a set of thresholds to suggest depositions. These thresholds are experimental for the moment.

```
name: Default
#--- Although other models can be build, there always has to be
#    a Default model
max-score: 10
ranges:
  - type: sloc
    #--- Valid types include:
    #    sloc:  Software Lines of Code
    #    raw:   Raw score
    #    bv:    Business value of app
    start: "0"
    end: int.max
    #--- describe a continuum of range bins
    ranges:
```

```
#--- Bin description
#    For any raw score between 0 and 100, Deploy to TAS, regardless
#    of buiness score
- type: raw #--- raw score bin range
  start: int.min #    start of raw bin
  end: "100" #    end of raw bin
  ranges:
    - type: bv #--- start of business value bin range
      start: flt.min #    start of bv range
      end: flt.max #    end of bv range
      outcome: #
        calculate: true
        #--- expression can be a complex formula that is based upon
        #    a combination of
        expression: max_score - log(10,raw_score)
        recommendation: Deploy to TAS

#--- Bin description
#    For any raw score between 101 and 100000
#    If BV less than 5, Rehost to TKG
#    If BV more than 5, Refactor to TAS
- type: raw
  start: "101"
  end: "10000"
  ranges:
    - type: bv
      start: flt.min
      end: "5.00"
      outcome:
        calculate: true
        expression: max_score - log(10,raw_score)
        recommendation: Rehost to TKG
    - type: bv
      start: "5.01"
      end: flt.max
      outcome:
        calculate: true
        expression: max_score - log(10,raw_score)
        recommendation: Refactor to TAS

#--- Bin description
#    For any raw score between 10001 and 10000000
#    If BV less than 5, Rehost to TKG
#    If BV more than 5, Refactor to TAS

- type: raw
  start: "10001"
  end: "10000000"
  ranges:
    - type: bv
      start: flt.min
      end: "5.00"
      outcome:
        calculate: true
```

```
          expression: max_score - log(10,raw_score)
          recommendation: Rehost to TKG
      - type: bv
        start: "5.01"
        end: flt.max
        outcome:
          calculate: true
          expression: max_score - log(10,raw_score)
          recommendation: Refactor to TAS

  #--- Bin description
  #    For any raw score greater than 10000001
  #    If BV less than 5, Rehost to TKG
  #    If BV more than 5, Refactor to TAS

  - type: raw
    start: "10000001"
    end: int.max
    ranges:
      - type: bv
        start: flt.min
        end: "5.00"
        outcome:
          calculate: true
          expression: max_score - log(10,raw_score)
          recommendation: Rehost to TKG
      - type: bv
        start: "5.01"
        end: flt.max
        outcome:
          calculate: true
          expression: max_score - log(10,raw_score)
          recommendation: Refactor to TAS
```

Adding rules

An important design requirement for `csa` was the ability to change rules in the field, without the need to recompile the executable. This requirement is driven by the realization that many customer may have in-house libraries that have `wrapper` classes and functions to simplify the use of other frameworks. As such, these wrapper classes may hide critical patterns. With this capability, those internal libraries can be scanned first and then the rules may be augmented to look for additional patterns. The following process details the steps required to do this.

1. Export the rules currently contained inside the `csa` executable

   ```
   csa rules export --output-dir=./rules
   ```

2. Add whatever rules you want to add to the rules directory that was just created. You may edit existing rules, as well.

3. Run the following commands

   To import rules, you'll need to ensure a `bins` directory is first exported. The following command will export a `bins.yaml` file to the default location of `~/csa-reports`. You only have to do this once unless you intend to modify or create new bins.

   ```
   csa bins export
   ```

You may then import rules with the following command.

```
csa rules import --rules-dir=./rules
```

All bins declared in the `bins.yaml` file are validated. As such, all tags specified in each `bin` must be present in the rules. If you intend to drop any or all of the standard rules, you'll need to remove bins from the `bins.yaml` file that depend on the rule tags.

4. Open a shell window (bash, MingW, git bash, powershell, etc…)
5. Run `csa help` with no parameters and you'll see usage instructions.
6. To target the current directory and analyze it simply type

   ```
   csa -p .
   ```

   The `-p` tell `csa` to treat each sub directory as a stand-alone application. Otherwise, the sub directories score will be rolled into a single application. In other words, the sub-directories are considered parts of a single application.
7. To target a directory with source code simple run

   `csa -p <path>` or `csa analyze -p <path>`

   In most usages it is expected that the user has `git` cloned multiple applications into a single directory and therefore each of those sub-directories is a single application.

Targeting an ear/war/jar

If no source code is available, you can decompile the ear, war, or jar files. To do so, you'll need the jar file `fernflower.jar` that is bundled in the `csa` download. We suggest putting the jar in the same directory as the `csa` executable, but this can be overridden with the `--fern-jar-path` flag.

1. Run `csa` and provide the fully qualified path to the "jar"

   ```
   csa analyze -p ~/resteasy-spring-2.3.8.Final-redhat-3.jar
   ```

Tool output

`csa` provides various useful outputs as it processes applications. These can be useful in understanding the results of the scan.

- Command `csa` is executing
- Critical directories
- Number of applications discovered
- Total files found in each application sub-directory
- Percent progress for each scan
- Software Lines of Code (SLOC) summary

While the scan occurs, `csa` is also loading a Sqlite database called `csa.db`. The default location is in the same directory in which the `csa` executable is located. When you first download `csa`, the `csa.db` does not exist, it is created the first time you run `csa`, your first run with be labeled `run 1` and each subsequent run will be incremented. You can have as many run's as you like in `csa.db` If you want to start with an empty `csa.db` you can delete or rename the file.

NOTE: If you download a new version of `csa` you will need to delete/rename the current `csa.db` to have any new rules appear in `csa`.

Rules

What is a Rule? A rule is in simplest terms a description of something that you want `csa` to detect. This description is structured so that `csa` can easily understand it but is designed to be flexible and extensible.

Important Note : Rules and analysis data are intentionally ephemeral from the perspective of `csa` and the baseline code. ***If you update rules, as discussed above, you'll want to keep them in some version controlled system, such as git. They represent valuable insight into your portfolio's profile.***

Understanding rules

Rule model

Attribute	Type	Description	Required (y/n)	Default	Overridable (y/n)
Name	string	The name of the rule. Can be meaningful or not but must be unique! And must match the name of the yaml file.	Y		N
FileType	string	The file extension the rule will target. I.E. `java` for `.java` files! Value should not include the dot (period). This can also be a regular expression. I.E. `xm[li]` would match both `xml` and `xmi` files	N	Rule will apply to all files if no value is specified	N
Container/Cloud	int	Factor to apply to effort score. This allows efforts to be scaled up or down. This is a percentage. 100, the default has no effect. 150 will raise the effort by 50%, 50 will lower the score by half.	N	100	N
Target	enum	This is the target of the rule. Valid values: File,Line. File = rule will apply to filenames only. Line = rule will be applied against every line of content within the file.	Y		N
Type	enum	This specifies the type or behavior of the rule. Valid values: regex, simple-text, simple-text-ci, starts-with, starts-with-ci, ends-with, ends-with-ci, contains, contains-ci	Y		Y
DefaultPattern	string	Pattern with a placeholder (%s) for substitution of "Pattern" values. I.E. "[.]%s[(]". This does not only apply to Regex rules but can also be used for others like a StartsWith such as 'org.json.%s'	N		Y (pattern)
Advice	string	Any advice on how to remediate this finding for cloud compatibility. This value is used if the specific pattern does not have advice.	N		Y
Score	int	A value indicating how this finding impacts cloud compatibility. At this time we have not settled on a scoring model so ...	N		Y
Category	string	The category of the rule. Simply a text marker to allow for grouping during analysis in csa. I.E. For the API rules this cotains the API name	N		N
Criticality	enum	A t-shirt size of the impact of the finding. Valid values: High, Medium, Low. Used for dashboard in csa	N		Y
Tags	array of Tag objects	Tags is a collection (0-n) of string values that can be used for grouping/slicing/ect... during analysis in csa	N		Y
Recipes	array of Recipe objects	Recipes is a collection (0-n) of URI values pointing at applicable recipes to aid in remediation of the finding	N		N

Attribute	Type	Description	Required (y/n)	Default	Overridable (y/n)
Patterns	array of Pattern objects	Patterns contains the patterns (1-n) that will be used to match against filenames/line data and result in findings	Y (at least 1)		N
ExcludePatterns	array of Pattern objects	Excludepatterns contains patterns that will be used to exclude false positives from findings. They will be applied to each findings and exclude the ones with positive match (Version 4.0 and higher)	N		N
Profiles	array of Tag objects	Profiles contains tags that will be used to filter rules that can be used during the scan. Rules can have multiple profile tags and profiles can be sepcified when running CSA ex: –profiles netcore,cloud-suitability (Version 4.0 and higher)	N		N

Pattern model

Attribute	Type	Description	Required (y/n)	Default
Value	string	This is the actual pattern value! It will be sustituted into or the default pattern or the overidding pattern.	Y	
Type	enum	This specifies the type or behavior of the pattern. Over-rides the rule type. Valid values: regex, simple-text, simple-text-ci, starts-with, starts-with-ci, ends-with, ends-with-ci, contains, contains-ci	Y	
Pattern	string	Pattern with a placeholder (%s) for substitution of Value. I.E. "[.]%s[(]". This does not only apply to Regex rules but can also be used for others like a StartsWith such as 'org.json.%s'	N	
Advice	string	Any advice on how to remediate this finding for cloud compatibility. Overrides any advice provided at the rule level.	N	
Score	int	A value indicating how this finding impacts cloud compatibility. At this time we have not settled on a scoring model so … Overrides any score provided at the rule level.	N	
Criticality	enum	A t-shirt size of the impact of the finding. Valid values: High, Medium, Low. Used for dashboard in csa. Overrides any Criticality provided at the rule level.	N	
Tags	array of Tag objects	Tags is a collection (0-n) of string values that can be used for grouping/slicing/ect… during analysis in csa. Overrides any tags provided at the rule level.	N	

Tag model

Attribute	Type	Description	Required (y/n)	Default
Value	string	the string you are tagging the rule or pattern with	N	

Recipe model

Attribute	Type	Description	Required (y/n)	Default
URI	string	A `uri` for the recipe to resolve the finding	N	

Example Rules (yaml)

Line level Regex This is the default annotations rule. It is only be applied against `.java` files, will be matched against every line in the file and detects this use of the annotations listed under patterns.

```yaml
name: annotations
filetype: java
target: line
type: regex
defaultpattern: ^.*@%s$
criticality: medium
tags:
  - value: annotations
patterns:
  - value: DeclareRoles
  - value: DenyAll
  - value: PermitAll
  - value: RolesAllowed
  - value: RunAs
  - value: Stateless
  - value: Stateful
  - value: MessageDriven
  - value: Entity
  - value: Init
  - value: Remove
  - value: ActivationConfigProperty
  - value: Local
  - value: Remote
  - value: LocalHome
  - value: RemoteHome
  - value: TransactionManagement
  - value: TransactionAttribute
  - value: PostActivate
  - value: PrePassivate
```

File Target This rule only gets applied against java files and detects the presence of pattern named under patterns section.

```yaml
name: java-iop
filetype: java$
target: line
type: regex
advice: Move to cloud friendly alternatives
defaultpattern: "^.*[ .]%s[ (.].*"
effort: 100
readiness: 6
category: iop
tags:
  - value: api
  - value: protocol
  - value: ejb
  - value: non-standard
patterns:
  - value: PortableRemoteObject
  - value: CodecFactory
  - value: CodecOperations
  - value: TransactionService
  - value: ServiceContext
  - value: TaggedComponent
  - value: TaggedProfile
```

Rules management

Exporting So, now you understand rules. What rules come by default? Or what do the current set of rules that `csa` is using look like? Let's export them!

Run the `csa` rules command with the export sub-command. By default rules will export to the default output-dir. You can override where they will go with the –output-dir flag or the –rules-dir flag. By default they will be exported as yaml with each rule in a separate file. If you prefer json or only working with a single file there are command flags to control this behavior. run `csa help rules export` for details.

```
usage: csa rules export [<flags>] [<name>]

export rule(s) from the database
```

```
==> csa rules export
DBEngine: sqlite Name: csa.db Version: 3.23.1
Successfully exported [47] rules @ [csa-reports/rules]
```

```
==> csa rules export
DBEngine: sqlite Name: csa.db Version: 3.23.1
Successfully exported [47] rules @ [csa-reports/rules]
```

Creating/Updating/Importing rules So, you can now see the rules that come by default. You want to change one, edit the file and update the rule. You want to create one, create a new file with the appropriate structure or add the rule to an existing file. Then run the `csa rules import` command. By default all rules in the –rules-dir will be imported or you can specify a rule name as an argument to the command. There are command flags to control directory where rules will be read and whether rules will be replaced or updated. Run `csa help rules import` for details.

```
usage: csa rules import [<flags>] [<name>]

import rule(s) into the database. By default rules will be added/updated rather than replace existing
```

```
==> csa rules import --rules-dir=csa-reports/rules
DBEngine: sqlite Name: csa.db Version: 3.23.1
Successfully imported [47] rule(s) found @[csa-reports/rules]
```

> **Note**: If importing more than one rule for file ==> If file format is yaml follow the standard yaml multi-document format of separating documents with `---`. If file format is json then just put the rule (object) in the file as a distinct object. Json really doesn't support more than one top level object in a file but that's ok! :). For example of how to create a multi-doc file run the export with the flag to create a single file and review!

Deleting/Removing You have a rule you don't want anymore. Or, for some reason, you want a clean slate…

Delete a rule

```
usage: csa rules delete <name>

delete a rule in the database
```

```
==> csa rules delete annotations
DBEngine: sqlite Name: csa.db Version: 3.23.1
Deleting rule [annotations]...done!
```

> **Note**: If the rule is found you will receive an indication it is deleted. If it is not found… you won't see any indication other than a clean exit(0) from the CLI

Delete All Rules (caution advised!)

```
usage: csa rules delete-all

delete all rules in the database!
```

```
==> csa rules delete-all
DBEngine: sqlite Name: csa.db Version: 3.23.1
Delete All Rules! Are you sure(y/n)? y
All Rules Successfully Deleted!
```

> **Note**: Rule 'filenames' are unimportant and have no bearing on rule behavior and are only important to the OS to disambiguate one file from another. Rule 'names' are only important from the perspective of they must be unique.

Rule Curation

CSA comes with hundreds of rules, but what most of us really need is a set of rules that describes what matters to us. Rule Curation is how we get there.

Rule Curation an iterative process for creating a set of rules that yield high value information in the context of your initiative.

Rule Curation usually involves two higher order phases of activity: 1. Reducing the vast ruleset that CSA ships with to a meaningful subset 2. Adding new rules for things CSA wasn't finding "out of the box"

In order to see the impact of our rule changes, we'll have to: 1. Import the modified ruleset 2. Rescan the portfolio

Investing a few weeks up front in Rule Curation is critical if you'll be driving decisions based on the scores CSA produces.

Each iteration of Rule Curation will impact scores. It's better to get the drastic swings in scores out of the way before decision-makers start incorporating scores into their decision-making.

Recalculate Scores After Rule Change CSA supports recalculating scores for prior runs without rescanning the code. This can be useful as part of the rule curation process described above.

Consider a scenario in which a portfolio scan yields a large grouping of applications with surprisingly low scores. Upon further inspection we see that all these applications are .NET applications using auto-generated machine keys, but only in their development profiles. Their production profiles are configured as expected.

In this case, we still want knowledge of where machineKeys are being autogenerated, but want to reduce the effort for the machine key rule to a much lower value, maybe even as low as zero; the rule is generating a lot of findings, and we know in this case each finding really is zero effort.

To get a clearer picture of our application scores in this case, we'll need to: 1. Export the rules from CSA 2. Lower the effort of the machine key rule to zero 3. Import the rules back into CSA 4. Scan the portfolio again to see the impact on scores

With the recalculate command we can recalculate the results of a prior run instead of rescanning the code with the updated rules.

```
csa recalculate --run-id=123 --alias="MachineKey Ruleset Change"
```

The above command will create a new run named "MachineKey Ruleset Change" with the findings from run 123 scored against the most recent rule import.

Rule Testing Framework

Creating or updating rules can be tedious. Working with regexes is always a challenge! How can you ensure that a rule will work as expected at runtime? Since CSA 4.0 a Rule testing framework has been added to help with adding tests along side any rule changes.

Test cases are written in YAML (No Go code required!) and can be managed independently of CSA. A new executable can be downloaded and used to test a set of rules locally. This is a great way for any organization using CSA and developing their own set of rules to ensure that rules are robust when used in production.

A new artifact is now available: rule-test.zip

It contains three main folders:

Attribute	Description
rules	Where rule YAML files should be stored
test-cases	Test cases written as YAML documents
test-samples	Sample of files that are leveraged by test cases
unit-test-l / unit-test-w / unit-test	Executable that contains all dependencies to run the unit tests

Create a Test Case

1. Find a sample of code to test the rule against

- Find one or multiple sample of code that the rule should be tested against
- Make sure to save the file under the right extension that matches the rule definition
- Place the file under /test-samples

2. Create the test case

- Add the test case to the proper test suite ex: cloud_blockers.yml for cloud blockers.

```
- Test Case Sample<br>
  tests:<br>
  - name: "Name of the test case"
    rule-name: rule-unique-name
    test-filename: test-sample-file-name.cs
    assert:  true or false (Is a match expected?)
    assert-count:  1 (How many matches expected?)
    assert-value: "null" or "Some code expected to be returned by the regex expression"
```

Run the Unit Test Suites

1. Navigate to the test directory

```
cd /test-rules
```

2. Run test command

On Linux:

```
WORK_DIR=$(pwd) ./unit-test-l -test.v
```

On MacOs:

```
WORK_DIR=$(pwd) ./unit-test -test.v
```

Publish Rule Updates

- Assuming the tests are passing (All the tests! To avoid regression), the changes can be pushed to the main repo referencing the related JIRA story

Application Archetypes

Bucketing of applications by tags

All rules in `csa` have any number of `tags` associated with them. A `tag` just associates a concept with the rule, such a `jni` or `corba`. At a higher level, we can think of a group of tags can further identify an architecture archetype or a bucket of similar applications. `csa` uses a single `yaml` file (`bins/bins.yaml`) to describe the archetypes as seen in the excerpt below:

```yaml
name: TKG
tags:
- name: Docker
  type: OR
- name: stateful
  type: AND
- name: javaee
  type: AND
- name: fullprofile
  type: AND
- name: jni
  type: OR
- name: nonstandard-protocol
  type: OR
- name: corba
  type: OR

---
name: TAS
tags:
- name: webprofile
  type: OR
- name: spring
  type: OR
- name: spring-boot
  type: OR
- name: webcontainer
  type: OR
- name: rest
  type: OR
- name: jar
  type: OR

.
.
.
```

CSA Web Interface

Overview

The `csa ui` command launches a browser-based visual explorer. You'll see `csa` write out some status information then the last line `Using Http FileSystem`. This is your indication that `csa ui` is waiting for you to directory your browser to `localhost:3001`

```
Csa: 1.63.0-rev.2 DBEngine: sqlite-3.25.2   DBName: csa.db
User: user
Command: ui
User-Home: /Users/user
DB Path: /Users/user/af/csa.db
Rules-Dir: /Users/user/csa/rules
OutputPath: /Users/user/csa/csa-reports
Exe Path: /Users/user/csa
Tmp Path: /var/folders/w6/3lp91tmn6b51wbqlzn2v6ywc0000gn/T/386908506

Using Http FileSystem!
```

We'll walk through all the pages of information available for your exploration. A note about features that are available throughout:

- For tabular data, most columns are searchable and sortable. You can also export the data to csv files for further analysis.
- Many of the graphics have a hover capbility that helps to identify more detailed information. This feature is very helpful when you have many applications on the Summary page scatter plot.

Summary Page

The Summary page is a high level view of your entire portfolio of applications. Notice the combo box in the upper left hand corner. If you have run several scans, each will be given a sequential run number starting at 1. `csa` always shows you it's latest run. You can select previous runs using thia control.

To the right of the combo box, there's a summary providing some information about the run you've selected.

The page selector has the summary page showing and highlighted in green. To select other pages, simple click on the one you want.

Just above the page selector you'll the see the current `csa` version number. Should you have problems always make sure you convey the version number to whomever you reach out to for help.

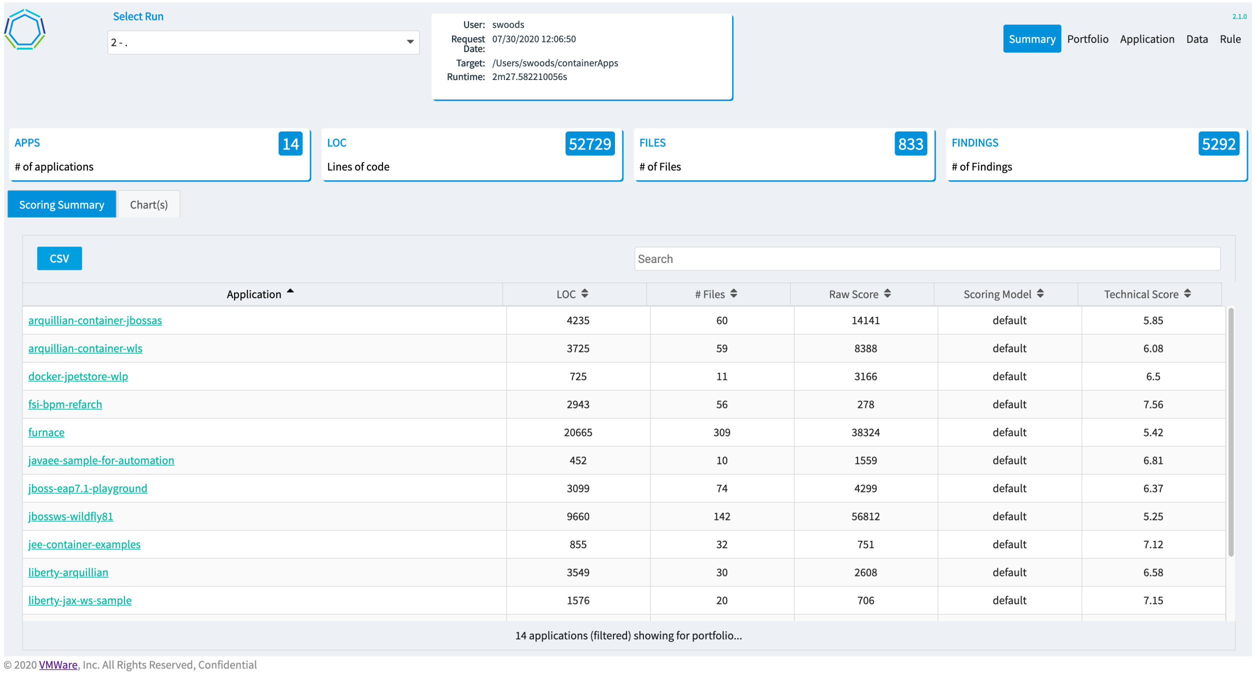
A series of infomation boxes divide the page showing high level statistics.

Box Title	Meaning
APPS	Total number of applications scanned
LOCS	Total lines of code found in portfolio
FILES	Total number of files in portfolio
FINDINGS	Total number of findings that were triggered by rules

The table below the information boxes has following columns:

Column	Description
Application	Name of directory or the application in the config file if there is one
LOC	Lines code source code
Files	Total number of files

Column	Description
Raw Score	Unadjusted score
Scoring Model	Currently there is only one `Default`
Technical Score	The overall calculated techncial score of the application



User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

APPS 14 # of applications
LOC 52729 Lines of code
FILES 833 # of Files
FINDINGS 5292 # of Findings

Application	LOC	# Files	Raw Score	Scoring Model	Technical Score
arquillian-container-jbossas	4235	60	14141	default	5.85
arquillian-container-wls	3725	59	8388	default	6.08
docker-jpetstore-wlp	725	11	3166	default	6.5
fsi-bpm-refarch	2943	56	278	default	7.56
furnace	20665	309	38324	default	5.42
javaee-sample-for-automation	452	10	1559	default	6.81
jboss-eap7.1-playground	3099	74	4299	default	6.37
jbossws-wildfly81	9660	142	56812	default	5.25
jee-container-examples	855	32	751	default	7.12
liberty-arquillian	3549	30	2608	default	6.58
liberty-jax-ws-sample	1576	20	706	default	7.15

14 applications (filtered) showing for portfolio...

© 2020 VMWare, Inc. All Rights Reserved, Confidential

Figure 1: enter image description here

Portfolio Page

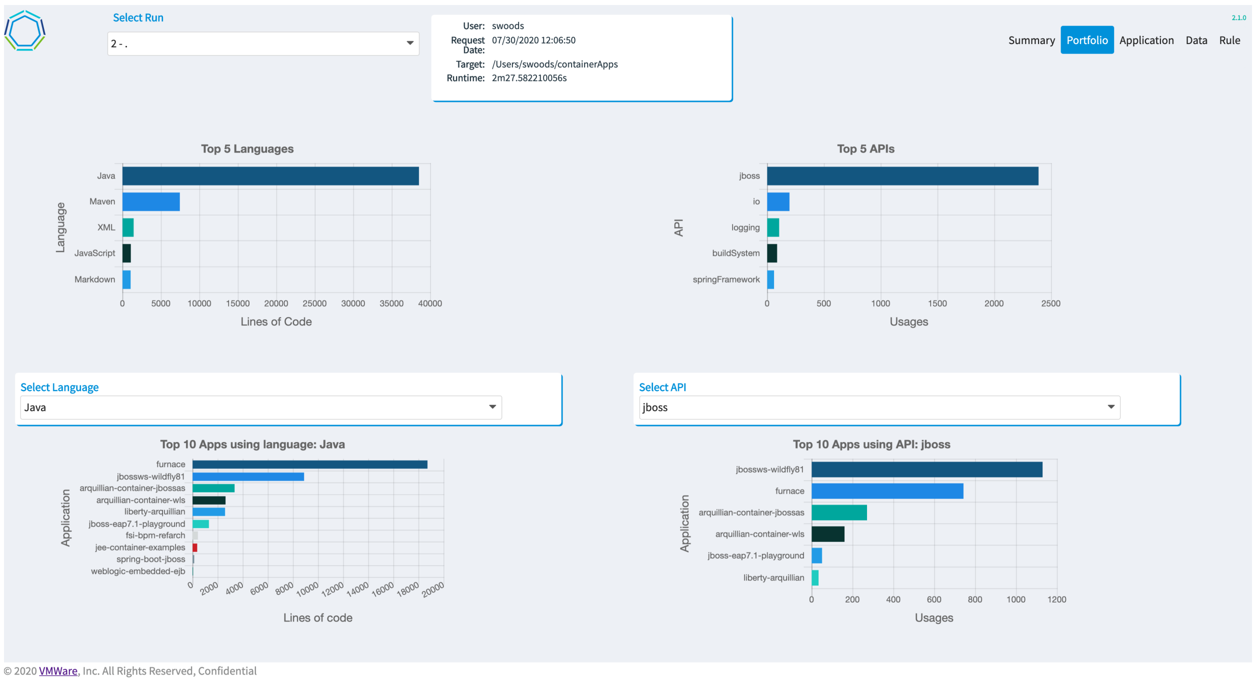
The Portfolio page gives a view of langages and APIs found across your portofolio. The top two bar charts present the top 5 languages and top 5 APIs found in your entire portfilio.

The bottom two bar charts present a selectable view with language and APIs, reflecting their presence, the top 10, found within each application.

Application Page

The Application page allows us to focus on a single application and dig deeper into its composition. Selecting an application in the combo box will show all the details for that application, such SLOC (Software Lines of Code) and number of files.

Box Name	Description
INFO	Rule findings that have a 0 score, they are informational only
LOW	Low effort score (1 - 3)
MEDIUM	Medium effort score (4 - 6)
HIGH	High effort score (7 - 10)
Total	Total number of finding
Score	Score for this application, hover to see the raw score



Select Run: 2 - .

User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

Select Language: Java

Select API: jboss

© 2020 VMWare, Inc. All Rights Reserved, Confidential

Figure 2: enter image description here

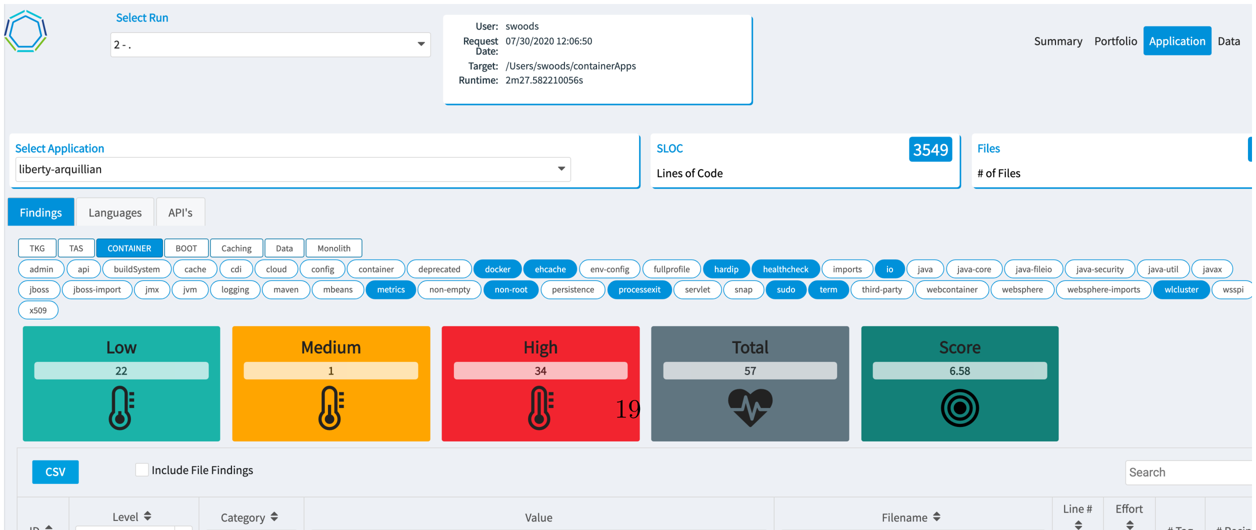
- Note: See scoring section for more detail. Ranges can change based upon the anticipated frequency of occurance.

Let's skip down to the table, then come back to the rectangles and ovals.

The tables contents are:

Column	Description
ID	Sequential ID number assocated with finding
Level	Level of effort score or information only
Category	Category of finding
Value	The pattern that triggers the rule, or a number indicating LOC, etc
File Name	File name where pattern was discovered
Line #	Line number where pattern occurred
Effort	Relative effort to remediate finding
# Tag	Total number of tags associated with finding
# Recipes	Total number of recipies associated with finding

- Note: `ID`, `# Tag`, and `# Recipes` are colored green, indicating more information available with a click.



Select Run: 2 - .

User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

Select Application: liberty-arquillian

SLOC 3549 Lines of Code

Files # of Files

Low 22

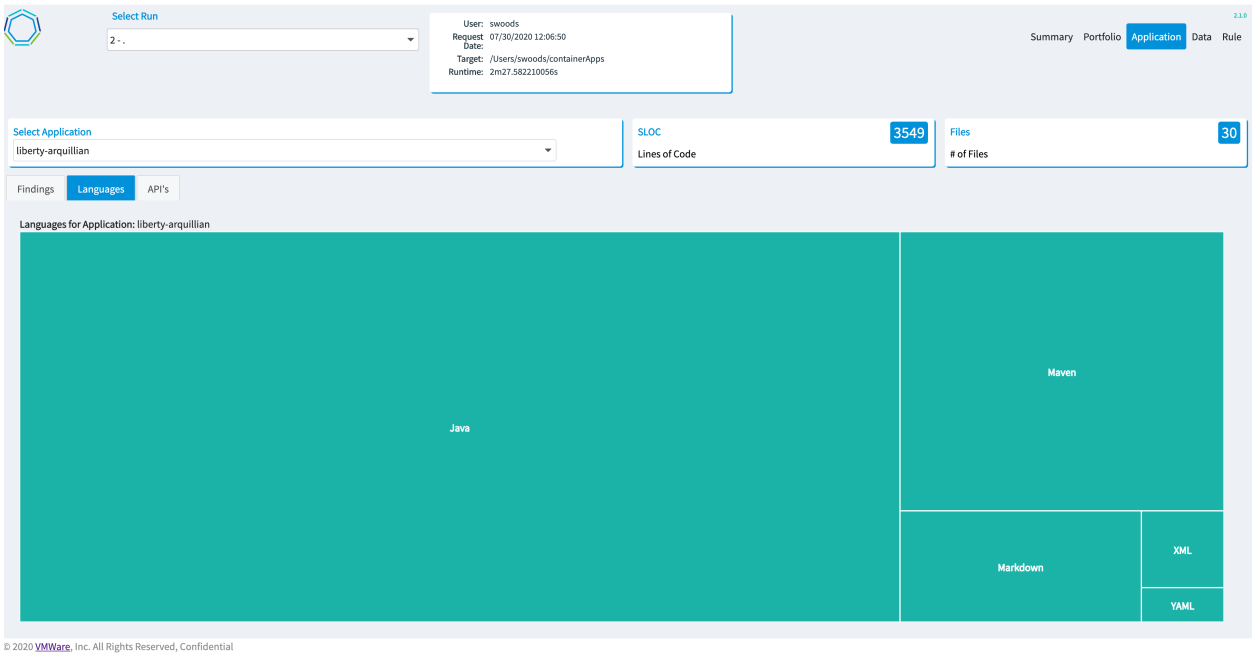
Medium 1

High 34

Total 57

Score 6.58

Include File Findings



Select Run: 2 - .

User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

Select Application: liberty-arquillian

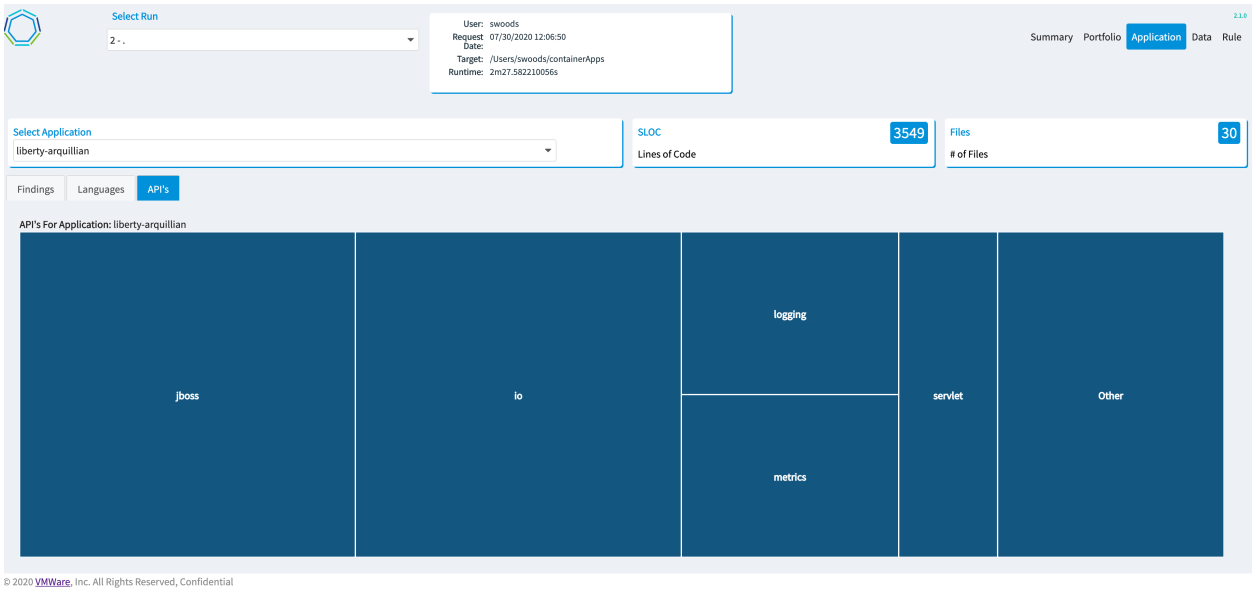
SLOC 3549 Lines of Code

Files 30 # of Files

Languages for Application: liberty-arquillian

© 2020 VMWare, Inc. All Rights Reserved, Confidential

Figure 4: enter image description here



Select Run: 2 - .

User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

Select Application: liberty-arquillian

SLOC 3549 Lines of Code

Files 30 # of Files

API's For Application: liberty-arquillian

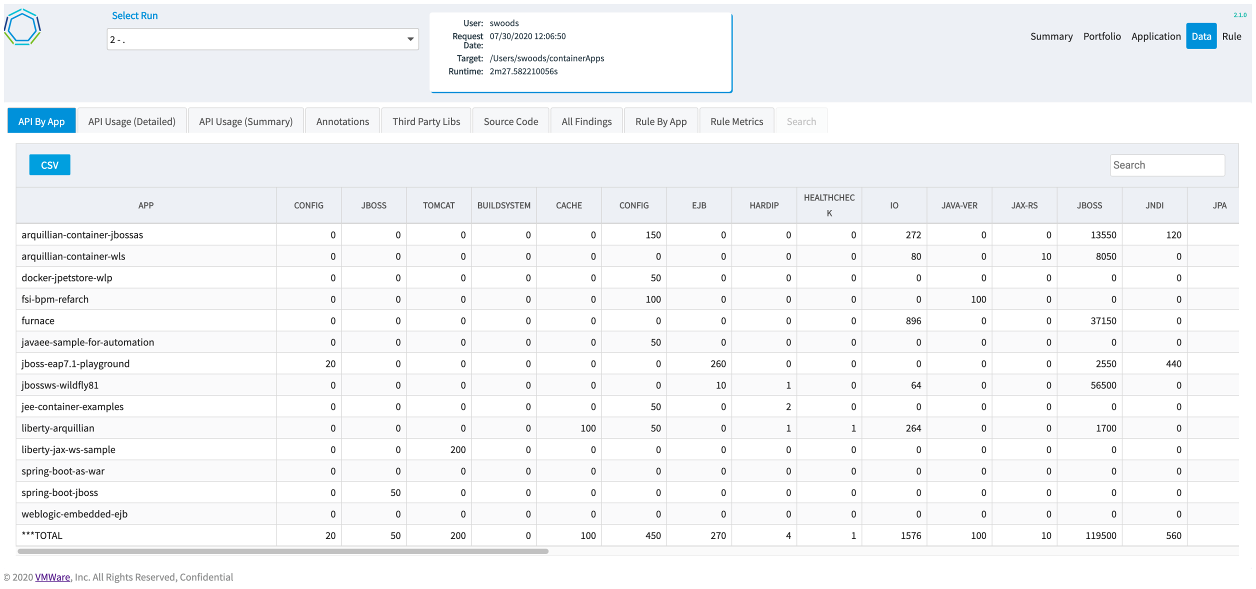
© 2020 VMWare, Inc. All Rights Reserved, Confidential

Figure 5: enter image description here

Data Page

The Data page provides several views into detailed findings for your entire portfolio. Using sorting and filtering you can explorer your application in several dimensions:

API by APP In `csa`, the primary component of scores is the individual rules that indicate patterns found in source code require remediation (`TAS`) or acommodation (`TKG`). While a single number score is useful in comparison with other applications, when considering a single application it is helpful to know the subcomponents of the score. This matrix shows the raw scores, with each column indicating an effort score. By horizontally scrolling you can better understand the score breakdown.



User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

APP	CONFIG	JBOSS	TOMCAT	BUILDSYSTEM	CACHE	CONFIG	EJB	HARDIP	HEALTHCHECK	IO	JAVA-VER	JAX-RS	JBOSS	JNDI	JPA
arquillian-container-jbossas	0	0	0	0	0	150	0	0	0	272	0	0	13550	120	
arquillian-container-wls	0	0	0	0	0	0	0	0	0	80	0	10	8050	0	
docker-jpetstore-wlp	0	0	0	0	0	50	0	0	0	0	0	0	0	0	
fsi-bpm-refarch	0	0	0	0	0	100	0	0	0	0	100	0	0	0	
furnace	0	0	0	0	0	0	0	0	0	896	0	0	37150	0	
javaee-sample-for-automation	0	0	0	0	0	50	0	0	0	0	0	0	0	0	
jboss-eap7.1-playground	20	0	0	0	0	0	260	0	0	0	0	0	2550	440	
jbossws-wildfly81	0	0	0	0	0	0	10	1	0	64	0	0	56500	0	
jee-container-examples	0	0	0	0	0	50	0	2	0	0	0	0	0	0	
liberty-arquillian	0	0	0	0	100	50	0	1	1	264	0	0	1700	0	
liberty-jax-ws-sample	0	0	200	0	0	0	0	0	0	0	0	0	0	0	
spring-boot-as-war	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
spring-boot-jboss	0	50	0	0	0	0	0	0	0	0	0	0	0	0	
weblogic-embedded-ejb	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
***TOTAL	20	50	200	0	100	450	270	4	1	1576	100	10	119500	560	

© 2020 VMWare, Inc. All Rights Reserved, Confidential

Figure 6: enter image description here

API Usage (Detailed) At a deeper level, the API Usage (Detailed) tab shows all details collected during the application scan. Using filters and sorting you can explorer all details of portfolio scans.

API Usage (Summary) The API Usage (Summary) presents a quick high level summary of the APIs found in your entire portfolio.

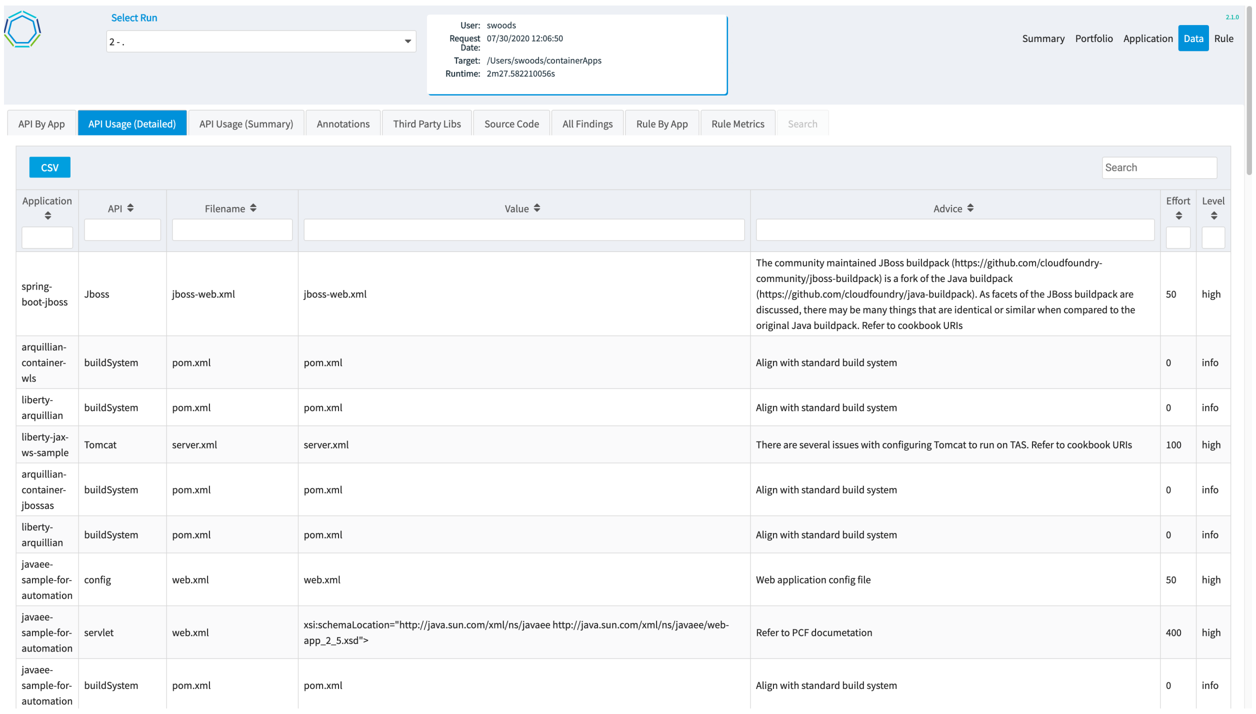
Annotations Here is collected all the annotation throughout your portfolio that may present challenges or considerations in your move to the cloud.

Third Party Libs Not all third party libraries behave well in the cloud, so it is helpful to know which are in your applications. They can be found here.

Source Code Sometimes, it's easy to forget how many languages are in your application. Here's is a breakdown of all those languages and the amount of each.

Rules Page

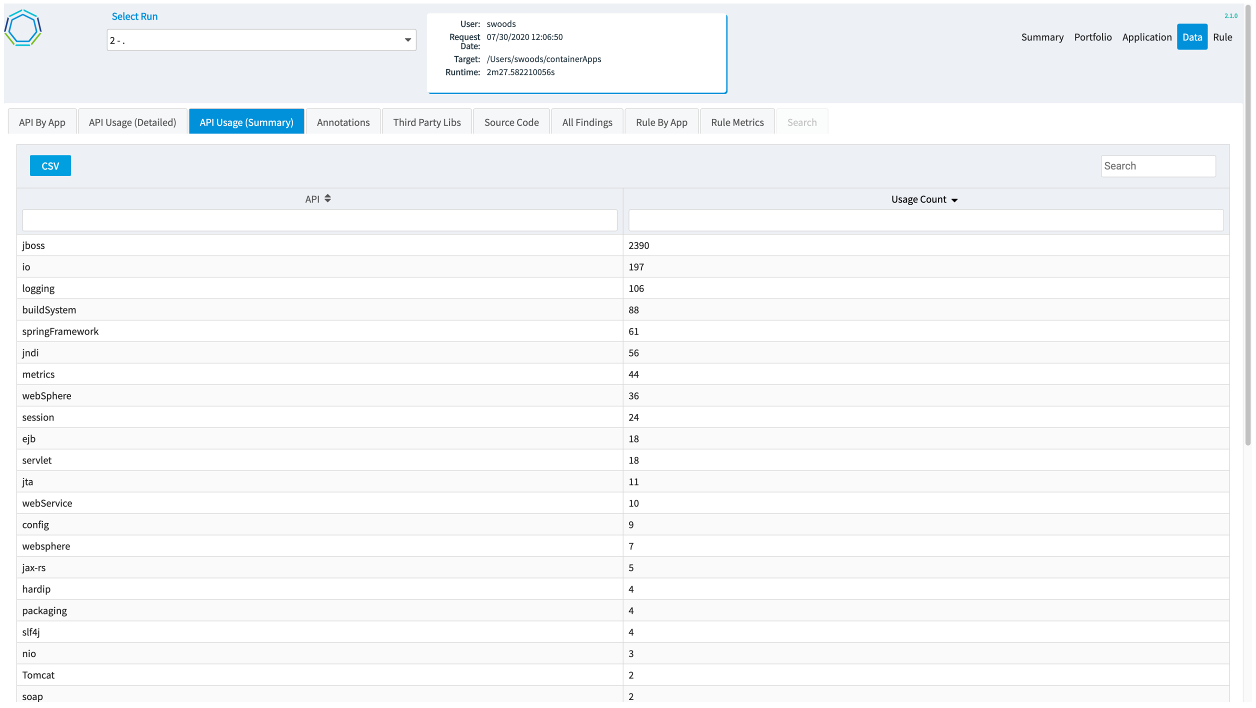
If you're writing rules for `csa` this tab is very helpful to know how the rules are performing. Poorly written regular expression can seriously affect `csa's` performance, this is where you to to find bottlenecks.



User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

Application	API	Filename	Value	Advice	Effort	Level
spring-boot-jboss	Jboss	jboss-web.xml	jboss-web.xml	The community maintained JBoss buildpack (https://github.com/cloudfoundry-community/jboss-buildpack) is a fork of the Java buildpack (https://github.com/cloudfoundry/java-buildpack). As facets of the JBoss buildpack are discussed, there may be many things that are identical or similar when compared to the original Java buildpack. Refer to cookbook URIs	50	high
arquillian-container-wls	buildSystem	pom.xml	pom.xml	Align with standard build system	0	info
liberty-arquillian	buildSystem	pom.xml	pom.xml	Align with standard build system	0	info
liberty-jax-ws-sample	Tomcat	server.xml	server.xml	There are several issues with configuring Tomcat to run on TAS. Refer to cookbook URIs	100	high
arquillian-container-jbossas	buildSystem	pom.xml	pom.xml	Align with standard build system	0	info
liberty-arquillian	buildSystem	pom.xml	pom.xml	Align with standard build system	0	info
javaee-sample-for-automation	config	web.xml	web.xml	Web application config file	50	high
javaee-sample-for-automation	servlet	web.xml	xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">	Refer to PCF documetation	400	high
javaee-sample-for-automation	buildSystem	pom.xml	pom.xml	Align with standard build system	0	info

Figure 7: enter image description here



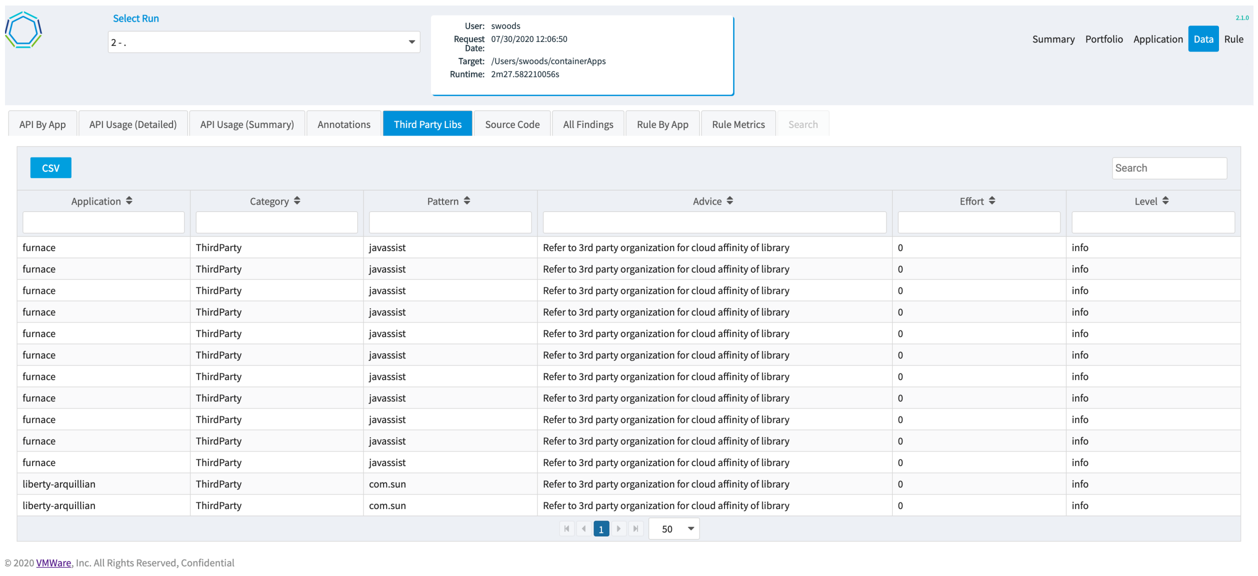
User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

API	Usage Count
jboss	2390
io	197
logging	106
buildSystem	88
springFramework	61
jndi	56
metrics	44
webSphere	36
session	24
ejb	18
servlet	18
jta	11
webService	10
config	9
websphere	7
jax-rs	5
hardip	4
packaging	4
slf4j	4
nio	3
Tomcat	2
soap	2

Figure 8: enter image description here

enter image description here

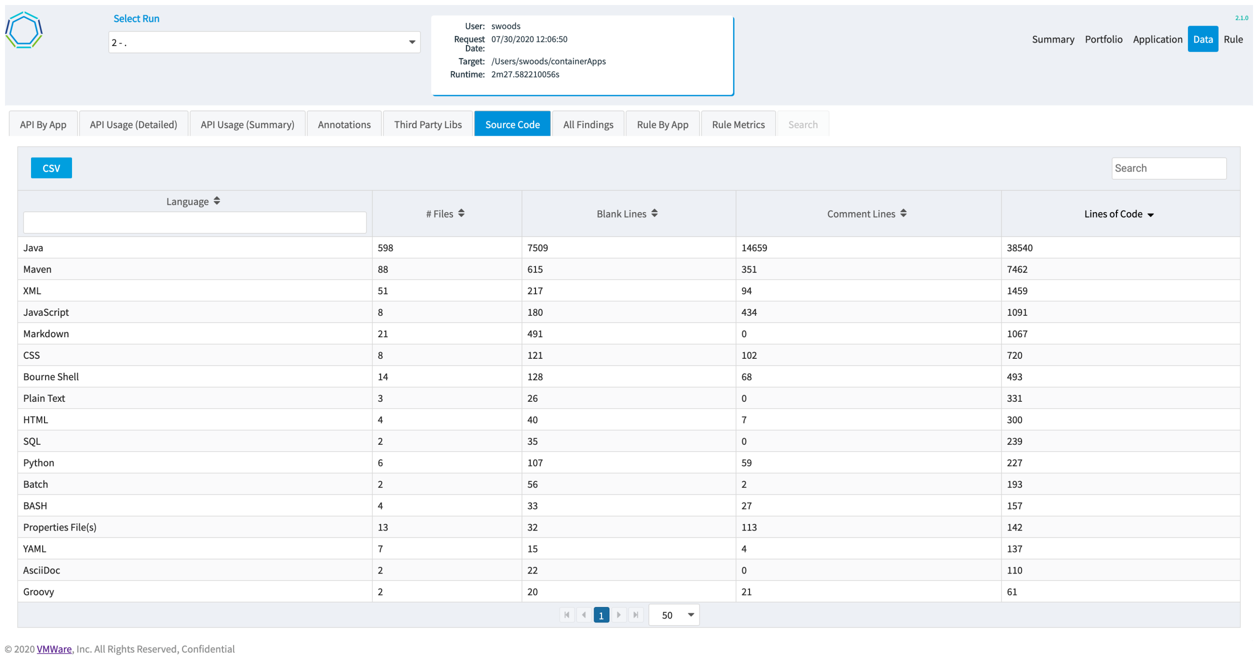
Figure 9: enter image description here



User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

Application	Category	Pattern	Advice	Effort	Level
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
furnace	ThirdParty	javassist	Refer to 3rd party organization for cloud affinity of library	0	info
liberty-arquillian	ThirdParty	com.sun	Refer to 3rd party organization for cloud affinity of library	0	info
liberty-arquillian	ThirdParty	com.sun	Refer to 3rd party organization for cloud affinity of library	0	info

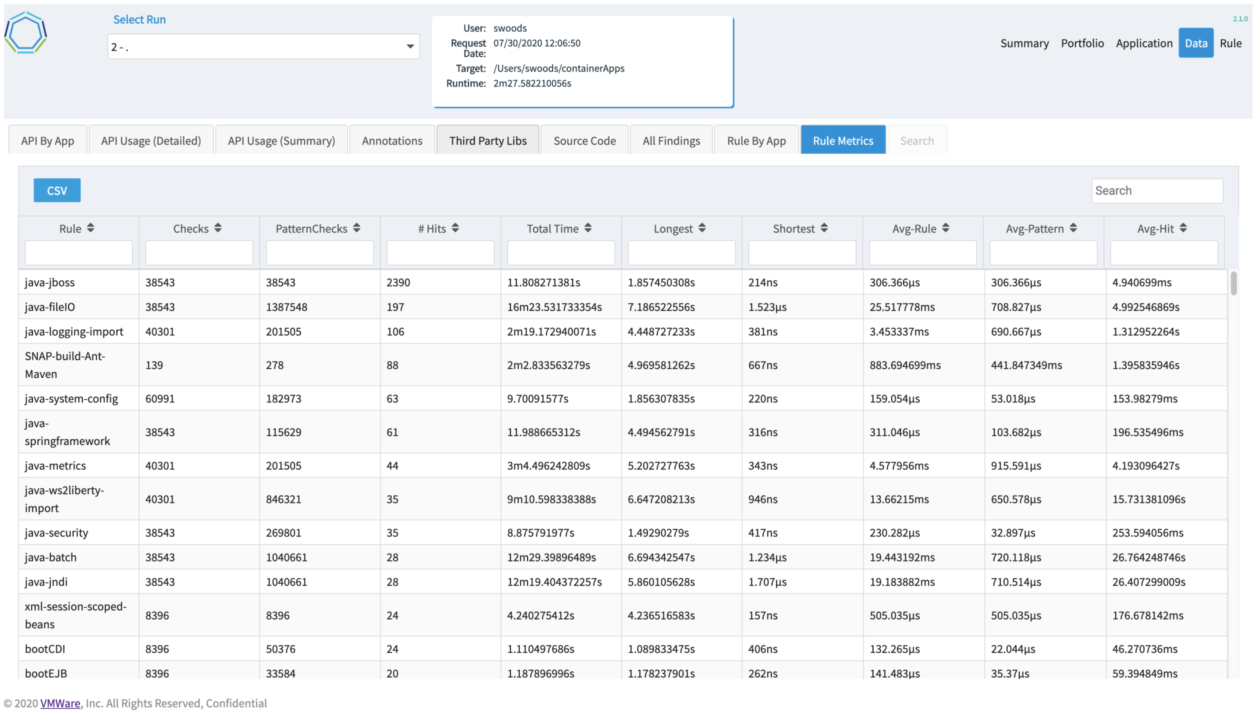
Figure 10: enter image description here



User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

Language	# Files	Blank Lines	Comment Lines	Lines of Code
Java	598	7509	14659	38540
Maven	88	615	351	7462
XML	51	217	94	1459
JavaScript	8	180	434	1091
Markdown	21	491	0	1067
CSS	8	121	102	720
Bourne Shell	14	128	68	493
Plain Text	3	26	0	331
HTML	4	40	7	300
SQL	2	35	0	239
Python	6	107	59	227
Batch	2	56	2	193
BASH	4	33	27	157
Properties File(s)	13	32	113	142
YAML	7	15	4	137
AsciiDoc	2	22	0	110
Groovy	2	20	21	61

Figure 11: enter image description here



User: swoods
Request Date: 07/30/2020 12:06:50
Target: /Users/swoods/containerApps
Runtime: 2m27.582210056s

Rule	Checks	PatternChecks	# Hits	Total Time	Longest	Shortest	Avg-Rule	Avg-Pattern	Avg-Hit
java-jboss	38543	38543	2390	11.808271381s	1.857450308s	214ns	306.366µs	306.366µs	4.940699ms
java-fileIO	38543	1387548	197	16m23.531733354s	7.186522556s	1.523µs	25.517778ms	708.827µs	4.992546869s
java-logging-import	40301	201505	106	2m19.172940071s	4.448727233s	381ns	3.453337ms	690.667µs	1.312952264s
SNAP-build-Ant-Maven	139	278	88	2m2.833563279s	4.969581262s	667ns	883.694699ms	441.847349ms	1.395835946s
java-system-config	60991	182973	63	9.70091577s	1.856307835s	220ns	159.054µs	53.018µs	153.98279ms
java-springframework	38543	115629	61	11.988665312s	4.494562791s	316ns	311.046µs	103.682µs	196.535496ms
java-metrics	40301	201505	44	3m4.496242809s	5.202727763s	343ns	4.577956ms	915.591µs	4.193096427s
java-ws2liberty-import	40301	846321	35	9m10.598338388s	6.647208213s	946ns	13.66215ms	650.578µs	15.731381096s
java-security	38543	269801	35	8.875791977s	1.49290279s	417ns	230.282µs	32.897µs	253.594056ms
java-batch	38543	1040661	28	12m29.39896489s	6.694342547s	1.234µs	19.443192ms	720.118µs	26.764248746s
java-jndi	38543	1040661	28	12m19.404372257s	5.860105628s	1.707µs	19.183882ms	710.514µs	26.407299009s
xml-session-scoped-beans	8396	8396	24	4.240275412s	4.236516583s	157ns	505.035µs	505.035µs	176.678142ms
bootCDI	8396	50376	24	1.110497686s	1.089833475s	406ns	132.265µs	22.044µs	46.270736ms
bootEJB	8396	33584	20	1.187896996s	1.178237901s	262ns	141.483µs	35.37µs	59.394849ms

© 2020 VMWare, Inc. All Rights Reserved, Confidential

Figure 12: enter image description here

Appendix A

CSA Structure and Operation

CSA is a complex application for several reasons. It contains a parallel rules processing engine, a web server, and an embedded set of yaml based rules. The first time it is run, it creates it's own *SQLite* database, `csa.db`. It's CLI syntax is extensive. To better understand it, a comprehensive call graph has been produced and may be found in the `doc/csa-callgraphs` directory. These call graphs were created with a `Go` base utility called go-callvis. The tools directs your browser to a locally hosted web-server.

To install:

```
go get -u github.com/ofabry/go-callvis
```

To run from the `csa/go` directory:

```
go-callvis csa.go
```

Otherwise, you open the `svg` files in the directory using any browswer.

Here's a reprint of the diagram legend:

Reference guide

Here you can find descriptions for various types of output.

Packages / Types

Represents	Style
`focused`	**blue** color
`stdlib`	**green** color

Represents	Style
`other`	**yellow** color

Functions / Methods

Represents	Style
`exported`	**bold** border
`unexported`	**normal** border
`anonymous`	**dotted** border

Calls

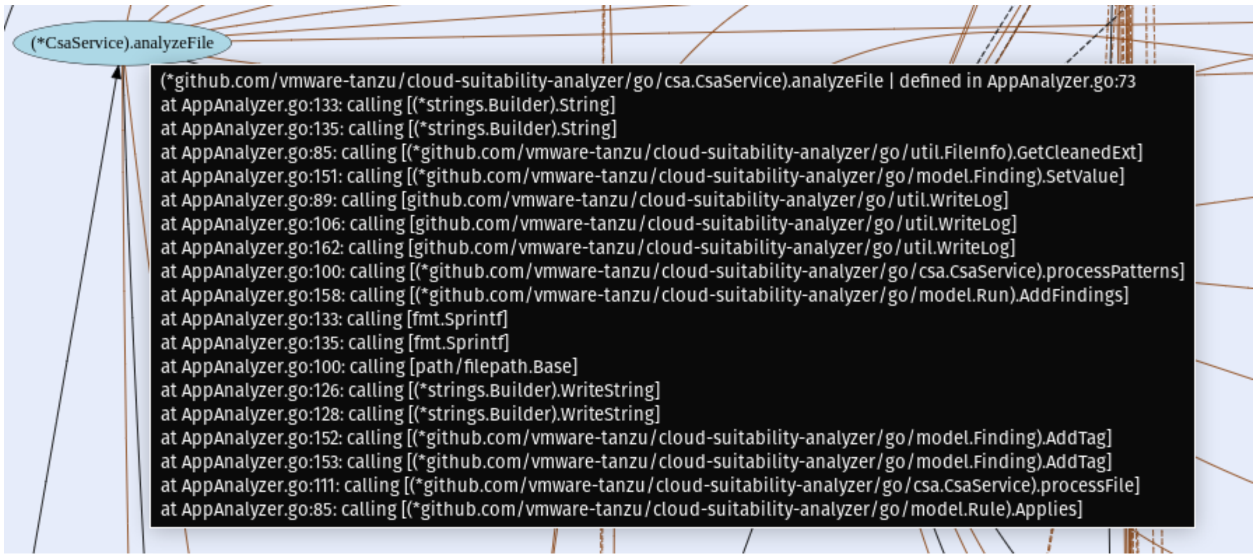
Represents	Style
`internal`	**black** color
`external`	**brown** color
`static`	**solid** line
`dynamic`	**dashed** line
`regular`	**simple** arrow
`concurrent`	arrow with **circle**
`deferred`	arrow with **diamond**

Package descriptions

Package	Description
`main.svg`	Main package
`csa.svg`	sloc
	app scoring
	natural lang processing
	lucene search
	git reports
`model.svg`	orm for sqlite
	rule validation
	yaml path
	xml query
	json path
	rule/file processor
	rule metrics
`routes.svg`	REST call routing
`db.svg`	ORM and data access layer
`profile.svg`	Runtime perf profiler
`kingpin.svg`	CLI parameter processer
`logrus.svg`	Logging framework
`natural.svg`	Natural Language processer
`reports.svg`	Git report generator
`search.svg`	Lucene search from CLI

A good first step if you are adding a feature to CSA is to grep the name of the function(s) in the `svg` files and determine which packages are involved then open the relevant `svg` to understand the implementation.

While exploring the `svg` in a browser, you will receive detailed source code information when hovering over a call graph node.



```
(*github.com/vmware-tanzu/cloud-suitability-analyzer/go/csa.CsaService).analyzeFile | defined in AppAnalyzer.go:73
at AppAnalyzer.go:133: calling [(*strings.Builder).String]
at AppAnalyzer.go:135: calling [(*strings.Builder).String]
at AppAnalyzer.go:85: calling [(*github.com/vmware-tanzu/cloud-suitability-analyzer/go/util.FileInfo).GetCleanedExt]
at AppAnalyzer.go:151: calling [(*github.com/vmware-tanzu/cloud-suitability-analyzer/go/model.Finding).SetValue]
at AppAnalyzer.go:89: calling [github.com/vmware-tanzu/cloud-suitability-analyzer/go/util.WriteLog]
at AppAnalyzer.go:106: calling [github.com/vmware-tanzu/cloud-suitability-analyzer/go/util.WriteLog]
at AppAnalyzer.go:162: calling [github.com/vmware-tanzu/cloud-suitability-analyzer/go/util.WriteLog]
at AppAnalyzer.go:100: calling [(*github.com/vmware-tanzu/cloud-suitability-analyzer/go/csa.CsaService).processPatterns]
at AppAnalyzer.go:158: calling [(*github.com/vmware-tanzu/cloud-suitability-analyzer/go/model.Run).AddFindings]
at AppAnalyzer.go:133: calling [fmt.Sprintf]
at AppAnalyzer.go:135: calling [fmt.Sprintf]
at AppAnalyzer.go:100: calling [path/filepath.Base]
at AppAnalyzer.go:126: calling [(*strings.Builder).WriteString]
at AppAnalyzer.go:128: calling [(*strings.Builder).WriteString]
at AppAnalyzer.go:152: calling [(*github.com/vmware-tanzu/cloud-suitability-analyzer/go/model.Finding).AddTag]
at AppAnalyzer.go:153: calling [(*github.com/vmware-tanzu/cloud-suitability-analyzer/go/model.Finding).AddTag]
at AppAnalyzer.go:111: calling [(*github.com/vmware-tanzu/cloud-suitability-analyzer/go/csa.CsaService).processFile]
at AppAnalyzer.go:85: calling [(*github.com/vmware-tanzu/cloud-suitability-analyzer/go/model.Rule).Applies]
```

Figure 13: Hover image