Odin - User Manual

ODIN

Student 1: James McDermott **Student Number:** 15398841

Student 2: Martynas Urbanavicius **Student Number:** 16485084

Project Supervisor: Dr. Stephen Blott

Completion Date: 16/05/2020

0. Table of Contents

6. Configuration

6.1 Odin Engine Users

6.2 The Odin Engine Config

6.3 Configuring the Data Store

6.4 Distribution

1. Dependencies

The Odin system is made up of several separate parts, each of which have respective dependencies. Each of these parts was developed using Go, Python or Node.js, the versions used are listed in section 1.1 below.

1.1 Programming Language and Build Tools

- Go
  - Version: 1.13
  - Usage: Odin Engine, Odin CLI, Odin Go SDK
- Python
  - Version: 3.7
  - Usage: Odin Python SDK
- Node.js -
  - Version: 10.16.2
  - Usage: Odin Observability Dashboard, Odin Node.js SDK
- GNU Make
  - Version: 4.1
  - Usage: Automating the local building and testing of the Odin Engine and Odin CLI

1.2 Odin Engine

- Chi
  - Version: v4.0.4
  - Description: A lightweight, idiomatic and composable router for building Go HTTP services.
  - Source Code: github.com/go-chi/chi

- Cronexpr
  - Version: v0.0.0
  - Description: A Golang cron expression parser.
  - Source Code: github.com/gorhill/cronexpr

- Raft
  - Version: v1.1.2
  - Description: A Golang implementation of the Raft consensus protocol.
  - Source Code: github.com/hashicorp/raft

- Logrus
  - Version: v1.5.0
  - Description: A structured and pluggable logging library for Go projects.
  - Source Code: github.com/sirupsen/logrus

- MongoDB Go Driver
  - Version: v1.3.1
  - Description: The MongoDB supported driver for Go.
  - Source Code: github.com/mongodb/mongo-go-driver

- Go Yaml
  - Version: v2.2.8
  - Description: A package that enables YAML support for Go.
  - Source Code: github.com/go-yaml/yaml

1.3 Odin CLI

- Cobra
  - Version: v0.0.6
  - Description: A library for creating powerful modern CLI applications.
  - Source Code: github.com/spf13/cobra

- Go Yaml
  - Version: v2.2.8
  - Description: A package that enables YAML support for Go.

- Source Code: github.com/go-yaml/yaml

1.4 Odin Software Development Kits

- Go
  - MongoDB Go Driver
    - Version: v1.3.1
    - Description: The MongoDB supported driver for Go.
    - Source Code: github.com/mongodb/mongo-go-driver
  - Go Yaml
    - Version: v2.2.8
    - Description: A package that enables YAML support for Go.
    - Source Code: github.com/go-yaml/yaml
- Python
  - Pymongo
    - Version: v3.10.1
    - Description: The MongoDB supported driver for Python.
    - Source Code: github.com/mongodb/mongo-python-driver
  - Ruamel.yaml
    - Version v0.15
    - Description: A library that enables YAML support for Python.
    - Source Code: sourceforge.net/projects/ruamel-yaml/
- Node.js
  - mongodb
    - Version ^3.5.6
    - Description: The official MongoDB driver for Node.js. Provides a high-level API on top of mongodb-core that is meant for end users.

        - Source Code: github.com/mongodb/node-mongodb-native
    - yamljs
        - Version ^0.3.0
        - Description: Standalone JavaScript YAML 1.2 Parser & Encoder. Works under node.js and all major browsers. Also brings command line YAML/JSON conversion tools.
        - Source Code: github.com/mongodb/node-mongodb-native

1.5 Odin Observability Dashboard

- Node.js

  Due to an extensive list of dependencies typical for web applications, the dependencies for the Odin Observability Dashboard can be found here:

  - src/webapp/client/package.json
  - src/webapp/server/package.json

2. Installation

2.1 Installing Languages, Compilers and Build Tools

2.1.1 Go

To install the Go compiler and associated tools on a general Linux system, run the commands below. Typically these commands must be run as root or through sudo.

```
wget https://dl.google.com/go/go1.13.10.linux-amd64.tar.gz

tar -C /usr/local -xzf go1.13.10.linux-amd64.tar.gz
```

Next add /usr/local/go/bin to the PATH environment variable using the following command. You can also do this by adding the command below to your /etc/profile (for a system-wide installation) or $HOME/.profile:

```
export PATH=$PATH:/usr/local/go/bin
```

Note: changes made to a profile file may not apply until the next time you log into your computer.

From here you should be able to access the Go command line tool by typing go. If the output should look like the below image then you've successfully installed Go!

```
Go is a tool for managing Go source code.

Usage:

    go command [arguments]

The commands are:

    build       compile packages and dependencies
    clean       remove object files and cached files
    doc         show documentation for package or symbol
    env         print Go environment information
    bug         start a bug report
    fix         update packages to use new APIs
    fmt         gofmt (reformat) package sources
    generate    generate Go files by processing source
    get         download and install packages and dependencies
    install     compile and install packages and dependencies
    list        list packages
    run         compile and run Go program
    test        test packages
    tool        run specified go tool
    version     print Go version
    vet         report likely mistakes in packages

Use "go help [command]" for more information about a command.
```

2.1.2 Node.js

To install Node.js and the associated package manager (npm) on a general Linux system, run the commands below. Typically these commands must be run as root or through sudo.

```
apt update

apt install nodejs
```

```
apt install npm
```

You can verify the installation of both nodejs and npm by running `node -v` and `npm -v` respectively.

2.1.3 Python3

To install Python3 and the associated package manager (pip) on a general Linux system, run the commands below. Typically these commands must be run as root or through sudo.

```
apt update

apt install python3

apt install python3-pip
```

You can verify the installation of both python3 and pip3 by running `python3 -V` and `pip3 -V` respectively.

2.1.4 Make

To install make on a general Linux system, run the commands below. Again these commands must be run as root or through sudo.

```
apt update

apt install make
```

2.2 Installing Odin Dependencies

2.2.1 Go

The installation of Go related Odin Engine and CLI dependencies has been automated through the use of a makefile in the src directory. Aside from this, if you wish to install the dependencies for the Go SDK you can use this command in the src/odin-libraries/go/odinlib directory:

```
go get -v ./...
```

To utilise the aforementioned automation, just run this command in the src directory:

```
make
```

This will perform a go get -v ./... operation on both the src/odin-cli and src/odin-engine directories. This will get all the dependencies the CLI and Engine relies on.

This make command also kicks off the build operation also. This will be described below in Section 3.

2.2.2 Node.js

If you wish to install the dependencies for the Odin Observability Dashboard and the Odin Node.js SDK, the run the following command:

`npm install`

In all of the following directories:

- src/webapp/client
- src/webapp/server
- src/odin-libraries/nodejs

2.2.3 Python

If you wish to install the dependencies for the Python SDK you can run this command in the `src/odin-libraries/python` directory:

```
pip3 install -r requirements.txt
```

3. Setup

First off, before building the project, we must first clone the repository via HTTPS with the following command:

```
git clone https://gitlab.computing.dcu.ie/mcdermj7/2020-ca400-urbanam2-mcdermj7.git
```

From here we move into the appropriate source code directory as specified below:

```
cd 2020-ca400-urbanam2-mcdermj7/src
```

To build the project, we can consult the `Makefile` in this directory. This file will automate the installation of the:

- Odin Engine
- Odin CLI
- MongoDB instance

Along with this, the Odin Engine will be run as a systemd service and the Odin CLI will be universally accessible from the `/bin` directory.

To utilise this automation we must run the makefile as the root user as so with the *make* command. This will:

- build the Odin Engine
- build the Odin CLI
- move the `odin-engine/config/odin-config.yml` file to the root user home directory
- move the generated CLI and Engine binary to the `/bin` directory
- move the `odin-engine/init/odin.service` file to `/lib/systemd/system` so it can be run as a systemd service
- install a locally accessible MongoDB
- creates the odin group, which users must be a member of to use the system.

We will see how the `odin-config.yml` file can be altered and configured to the users needs in Section 6.1

We can verify all components were successfully install with the following series of commands:

```
systemctl status odin

systemctl status mongod

odin --help
```

It is advised the first two commands are run through root or with sudo. These commands will allow systemd to report the current status of the Odin Engine and the local MongoDB instance.

The final command will verify you have a working install of the Odin CLI tool.

It's also advisable to set the following two environment variables for the Odin Engine to use:

```
export ODIN_EXEC_ENV=True
export ODIN_MONGODB="mongodb://localhost:27017"
```

The purpose of these environment variables and their configuration is further detailed in section 6 of this document.

To set up the Odin Observability Dashboard web application first follow Node.js installation in section 2.1.2 as well as installing the project dependencies outlined in in section 2.2.2 thereafter.

Odin Observability Dashboard web application backend server setup :

Take time to edit the `.env` file found in the `src/webapp/server` directory to include your MongoDB connection URL.

Then, the `src/webapp/server` directory, run `npm start` to start up the backend server. This server will start listing on port 3000 and will be accessible at `http://localhost:3000`.

Odin Observability Dashboard web application frontend setup :

In the `src/webapp/client` directory, you can run `npm install -g @angular/cli@latest` to install the latest version of the Angular CLI tool

Then, run `ng serve` in the same directory to start the frontend on port 4200, which will make the user interface accessible at `http://localhost:4200`

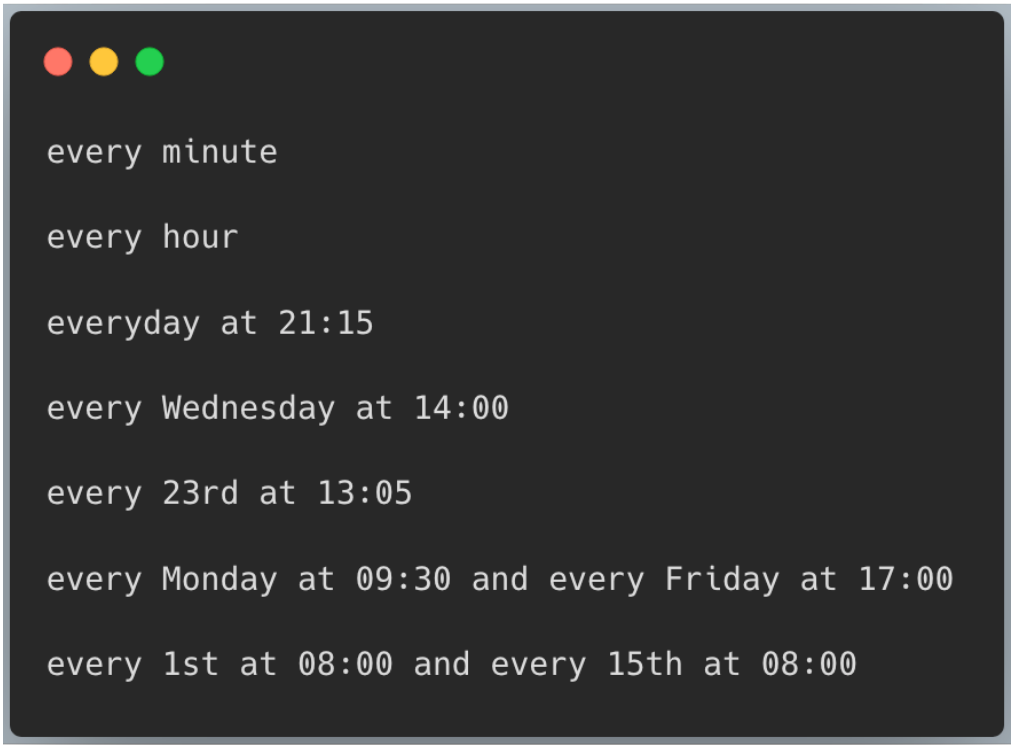
4. Usage

4.1 The Odin Scheduling Format

The Odin Scheduling Format is heavily reliant on the simplicity of the English language.

Rather than fumble around with the cron schedule string syntax, we have created a more human readable scheduling format that leverages a subset of specific keywords to form a robust schedule string syntax.

Here are some examples of acceptable Odin schedule strings:



4.2 The Odin CLI

4.2.1 Getting help

To view all available Odin CLI commands you can simply run:

```
odin --help
```

To view available flags and recommended usage of a specific command, run:

`odin <command_name> --help`

4.2.2 Generate required job files

Every Odin job is defined by two separate files:

- The user code (a .py, .go or .js file) used to define a jobs operation(s)
- Supplementary YAML (a .yml or .yaml file) which contains associated job metadata

We can generate the appropriate files for our job like so:

`odin generate -f amzn_stocks.yml -l python`

-f specifies a new YAML file for the job in the format: `<jobName>.yml`

The <jobName> passed here will also be used to name the user code in the form of `<jobName>.py`, `<jobName>.go` or `<jobName>.js`

-l defines the language used for the job. The acceptable values for the *-l* parameter are:

- python
- go
- node

In the first example we generated two files:

- amzn_stocks.yml
- amzn_stocks.py

`odin generate -f backup.yml -l go`

Similarly in this second example we generate two files again, but this time the extension of the user code file is different:

- backup.yml
- backup.go

The contents of the YAML file are purposely bare in parts:

```yaml
provider:
  name: 'odin'
  version: '1.0.0'
job:
  id: 'e791249cec6a'
  name: ''
  description: ''
  language: 'go'
  file: 'backup.go'
  schedule: ''
```

We leave the completion of the name, description and schedule fields to the user! The name and descriptions can be set at the user's discretion, and we advise you to consult the Odin scheduling format in section 4.1 of this document when setting the schedule string in the YAML file.

The generated user code file is purposely left completely empty for the user to fill their code in.

4.2.3 Deploying a job

Once you have generated your job files and filled them out as appropriate, deploying your job is simple. Here's a look at the files used during this example deployment.

YAML: amzn_stocks.yml

```yaml
provider:
  name: 'odin'
  version: '1.0.0'
job:
  id: '6d7341fa3fea'
  name: 'amzn'
  description: 'Check amazon stock price'
  language: 'python3'
  file: 'amzn_stock.py'
  schedule: 'every minute'
```

USER CODE: amzn_stocks.py

```python
import requests
from bs4 import BeautifulSoup as bs
import pyodin

odin = pyodin.Odin(config="amzn_stock.yml")
r = requests.get('https://finance.yahoo.com/quote/AMZN?p=AMZN')

if odin.condition("check status code", r.status_code == 200):
    soup = bs(r.content, 'lxml')
    for stock in soup.find_all('span', class_='Trsdu(0.3s) Trsdu(0.3s) Fw(b) Fz(36px) Mb(-4px) D(b)'):
        odin.watch("current price", stock.get_text())
        odin.result("success", "200")
else:
    odin.result("failure", "500")

```

We simply deploy the job with the following command:

```
odin deploy -f amzn_stocks.yml
```

Where *-f* specifies the YAML file for the job.

We can check if this deployment was successful by running:

```
odin list
```

Running this command gives us the following output:

We can see that we have successfully deployed our job! The deploy command is aliased to both dep and add. The list command is aliased to ls.

4.2.4 Removing a job

We can remove any job by fetching it's ID displayed with odin list and using using it in the following command:

```
odin remove -i a37b187f5930
```

We can check if this removal was successful by running odin list again:

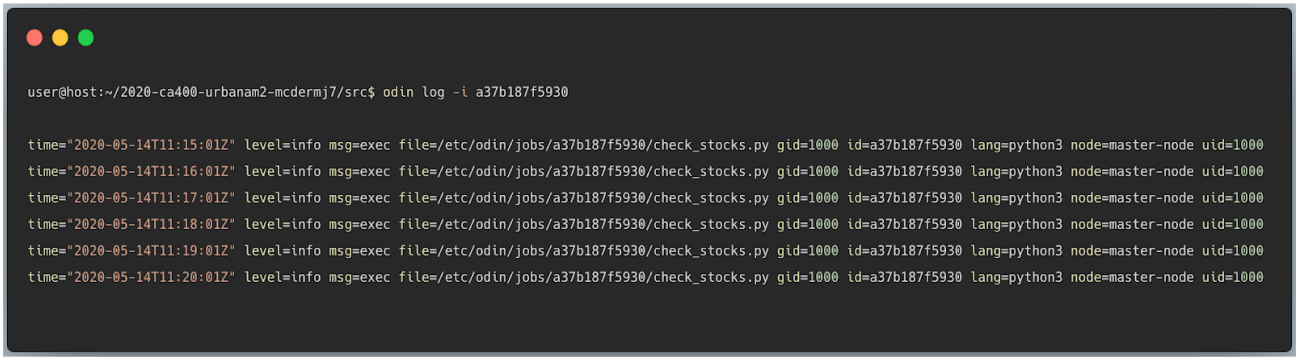
We can see that we have successfully removed that job! The remove command is aliased to rm.

4.2.6 Show a jobs log

We can view the execution logs of any job by fetching it's ID displayed with odin list and using using it in the following command:

odin log -i a37b187f5930

This will yield an output which looks like this:



In the above image we can see:

- the time of execution
- the message level (info or warning)
- the message attached to this log (exec or failed)
- the user code of the job that was executed
- the uid and gid of the user who deployed that job
- the language of the job executed
- the node on which it the job was executed

4.2.7 Describe running jobs

We can view the execution logs of any job by fetching it's ID displayed with `odin list` and using using it in the following command:

`odin describe -i a37b187f5930`

This will yield an output which looks like this:

We can see that we have successfully viewed that job description! The `describe` command is aliased to `desc`.

4.2.8 Modify running jobs

We can modify a jobs name, description and schedule with the `odin modify` command. Once more we must fetch the ID to specify the job we wish to modify. Once you have the ID you can change the name, description and schedule with the `-n`, `-d` and `-s` flags respectively. Below are some examples of the odin modify command.

Modify the description alone:

```
odin modify -i a37b187f5930 -d "check my stocks"
```

Modify the schedule alone:

```
odin modify -i a37b187f5930 -s "every hour"
```

Modify the name alone:

```
odin modify -i a37b187f5930 -n "stock_check"
```

Modify the description and the schedule:

```
odin modify -i a37b187f5930 -d "a job to check my stocks" -s “everyday at 18:00”
```

As long as you remember to include quotation marks around the new value for any of the values you want to change, the modify command should work and you can view your successful changes with the `odin list` command once more.The `modify` command is aliased to `mod`.

4.2.9 Link jobs together

When you run the `odin list` command you will see a LINKS heading, which is empty by default. Links in Odin reference a concept where the successful execution of one job will in turn call another job.

Let's say the `amzn_stock` job runs every hour and stores the information in a file in the user home directory. If this job executes successfully we would like it to call another job, `email_file`.

The schedule `never` is totally acceptable in the Odin Engine - it essentially denotes that a job will never execute by itself but will only execute if linked to another job.

Once we deploy this job running `odin list` will give us this output:

We can link the stock check job to the email job with the following command:

```
odin link -f a37b187f5930 -t f0aedde327f3
```

Running `odin list` will show us that the link between these jobs now exists:

ID	NAME	LANGUAGE	LINKS	SCHEDULE
f0aedde327f3	email_file	go		never
a37b187f5930	check_stocks	python3	f0aedde327f3	Every hour

4.2.10 Unlink jobs from each other

We can undo the work done by the `odin link` command using the unlink command like so:

`odin unlink -f a37b187f5930 -t f0aedde327f3`

This decouples the two jobs into independent tasks once more.

4.2.11 Recover job files

While working, let's say you accidentally removed the user code and/or the YAML file. There's no need to fret, the Odin Engine has already stored them for you in case this happens, and it will continue to execute them for you.

If you need to recover these files for any reason you can fetch the ID by using `odin list` and use it in the command below:

```
odin recover -i a37b187f5930
```

Running `ls` in your current directory will show that the files have been fully restored.

4.2.12 Add more Odin Engine nodes

The Odin Engine leverages the Raft consensus protocol to run as a distributed system. Distributed systems offer reliability to systems in case of a failure, and Odin proves to be an easily scalable and flexible system.

By default, the initial Odin Engine node is called “master-node”. If you want to add nodes to the cluster you can do so simply with the command:

```
odin nodes add -n worker1 -a :39391 -r :12001
```

Breaking down this command we see that:

- The name (-n) of the new worker node is `worker1`
- The http address port provided for the new worker is `:39391`
- The raft port provided for the new worker is `:12001`

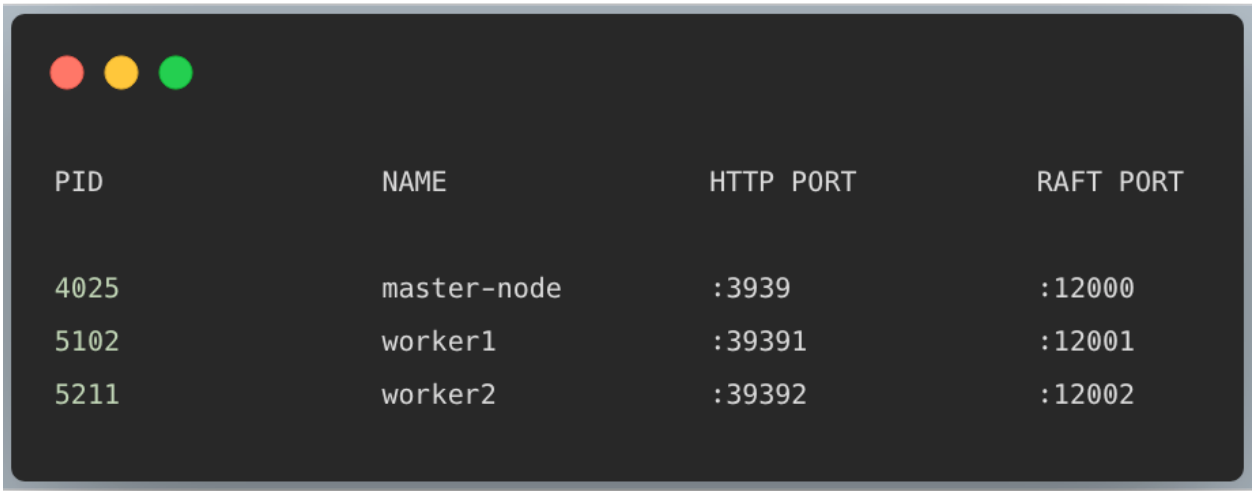
We can add another node like so:

`odin nodes add -n worker2 -a :39392 -r :12002`

We can verify the addition of this new nodes with the following command:

```
odin nodes get
```

Which returns a list of nodes in the cluster:



PID	NAME	HTTP PORT	RAFT PORT
4025	master-node	:3939	:12000
5102	worker1	:39391	:12001
5211	worker2	:39392	:12002

In regards to distributed systems, in particular with raft based systems, it's advisable to run 3-node clusters or 5-node clusters. This is recommended as:

- 3-node clusters can tolerate a failure in 1 node
- 5-node clusters can tolerate a failure in any 2 nodes

Please consult section 4.4 in regards to further details on running Odin as a distributed system.

4.2.13 View Job Stats

You can directly view statistics with the `odin stats` command by specifying the ID of the job you wish to view. In the case of the earlier `amzn_stock.yml` job we can type:

```
odin stats -i 6d7341fa3fea
```

This gives us an an output which will look something like this:

```
ID              DESCRIPTION         TYPE              VALUE
6d7341fa3fea    successful request  condition         True
6d7341fa3fea    current price       watch             2409.78
6d7341fa3fea    success             result            200
```

These values from the code are generally tracked thanks to the Odin Software Development kits. You can learn how to implement these in the next section!

4.3 The Odin Software Development Kits

In each of the following three subsections we will demonstrate how a language specific SDK is used in conjunction with that language. Operationally each section is the exact same.

4.3.1 Python SDK

The Python SDK can be imported as a pip package using the following command:

```
pip3 install pyodin
```

You can create the odin object in your python user code like so:

```
import pyodin
odin = pyodin.Odin(config="your_config_here.yml")
```

It's important you specify the name of your YAML configuration file here as the Python SDK utilises metadata from it.

From here it's quite simple, you have three distinctive operations:

- Watch
- Condition
- Result

Let's look at the watch operation with respect to the following code snippet:

```
1 import requests
2 import pyodin
3
4 odin = pyodin.Odin(config="your_config_here.yml")
5 session = requests.Session()
6 response = session.get("https://en.wikipedia.org/wiki/Special:Random")
7 odin.watch("url fetched", response.url)
```

This snippet fetches a random url from wikipedia on each execution. The watch operation stores this url along with a string "url fetched" to annotate the variable being stored.

This allows us to better debug our jobs for when they don't work as intended. If this job failed, we could immediately diagnose whether or not it was because a url wasn't generated from line 6.

Let's now look at this code again, but with the condition operation added:

```
1 import requests
2 import pyodin
3
4 odin = pyodin.Odin(config="your_config_here.yml")
5 session = requests.Session()
6 response = session.get("https://en.wikipedia.org/wiki/Special:Random")
7
8 if odin.condition("check if url is empty", response.url != ""):
9     odin.watch(“url fetched”, response.url)
```

This time, we are introducing a new line at line 8. This `condition` operation will store the boolean equivalent to the statement `response.url != ""` and will be annotated by the string “check if url is empty”.

If the statement `response.url != ""` is true then the `condition` operation will return true and progress to the `watch` operation on line 9.

Finally, we take a look at the `result` operation:

```python
import requests
import pyodin

odin = pyodin.Odin(config="your_config_here.yml")
session = requests.Session()
response = session.get("https://en.wikipedia.org/wiki/Special:Random")

if odin.condition("check if url is empty", response.url != ""):
    odin.watch(“url fetched”, response.url)
    odin.result("fetched successfully", 0)
else:
    odin.result("failure when fetching", 1)
```

This operation acts like a return statement, once it’s executed the job is considered to be over. A successful attempt is denoted by a 0 while a failed attempt is denoted by a 1.

In either case, the result is annotated by a string once more. Once a result operation is run, the code will finish execution.

4.3.2 Go SDK

The Go SDK can be imported using the following command:

```
go get gitlab.computing.dcu.ie/mcdermj7/2020-ca400-urbanam2-mcdermj7/src/odin-libraries/go/odinlib
```

You can create the odin entry point in your go user code like so:

```
package main
```

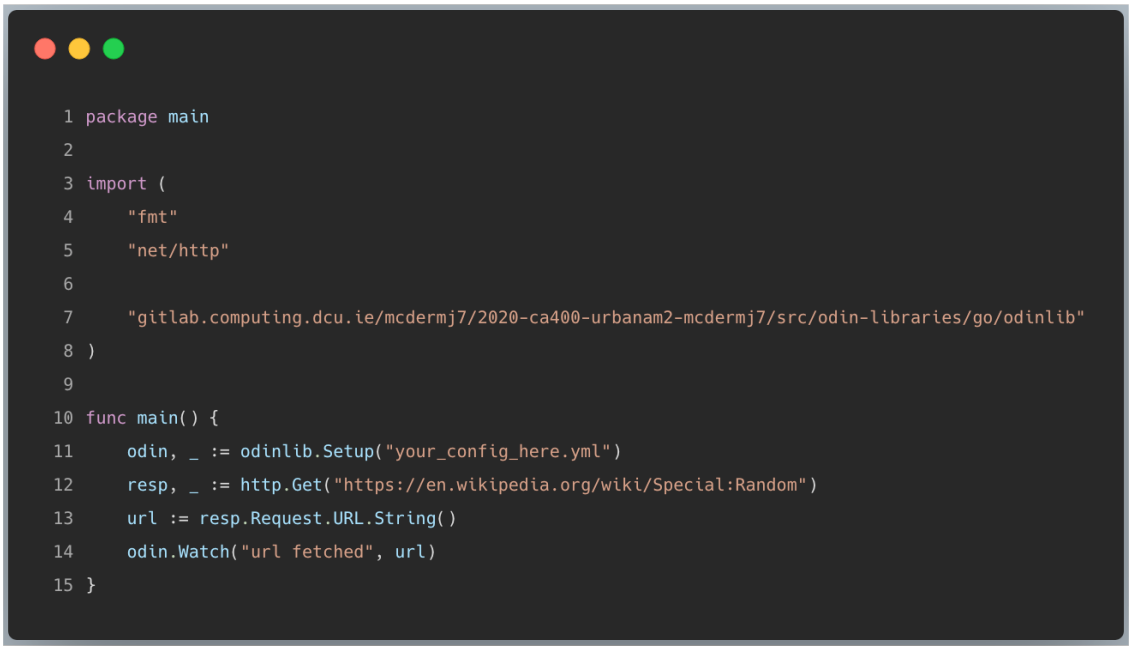
```
import (
    "gitlab.computing.dcu.ie/mcdermj7/2020-ca400-urbanam2-mcdermj7/src/odin-libraries/go/odinlib"
)

func main() {
    odin, _ := odinlib.Setup("your_config_here.yml")
}
```

It's important you specify the name of your YAML configuration file here as the Go SDK utilises metadata from it.

From here it's quite simple, you have three distinctive operations:

- Watch
- Condition
- Result

Let's look at the `Watch` operation with respect to the following code snippet:



```go
package main

import (
    "fmt"
    "net/http"

    "gitlab.computing.dcu.ie/mcdermj7/2020-ca400-urbanam2-mcdermj7/src/odin-libraries/go/odinlib"
)

func main() {
    odin, _ := odinlib.Setup("your_config_here.yml")
    resp, _ := http.Get("https://en.wikipedia.org/wiki/Special:Random")
    url := resp.Request.URL.String()
    odin.Watch("url fetched", url)
}
```

This snippet fetches a random url from wikipedia on each execution.

The watch operation stores this url along with a string “url fetched” to annotate the variable being stored.

This allows us to better debug our jobs for when they don’t work as intended. If this job failed, we could immediately diagnose whether or not it was because a url wasn’t generated from line 6.

Let’s now look at this code again, but with the `Condition` operation added:

```
1  package main
2
3  import (
4      "fmt"
5      "net/http"
6
7      "gitlab.computing.dcu.ie/mcdermj7/2020-ca400-urbanam2-mcdermj7/src/odin-libraries/go/odinlib"
8  )
9
10 func main() {
11     odin, _ := odinlib.Setup("your_config_here.yml")
12     resp, _ := http.Get("https://en.wikipedia.org/wiki/Special:Random")
13     url := resp.Request.URL.String()
14     if odin.Condition("check if url is empty", url != "") {
15         odin.Watch("url fetched", url)
16     }
17 }
```

This time, we are introducing a new line at line 8. This `Condition` operation will store the boolean equivalent to the statement `url != ""` and will be annotated by the string “check if url is empty”.

If the statement `url != ""` is true then the `Condition` operation will return true and progress to the `Watch` operation on line 9.

Finally, we take a look at the `Result` operation:

```go
package main

import (
    "fmt"
    "net/http"

    "gitlab.computing.dcu.ie/mcdermj7/2020-ca400-urbanam2-mcdermj7/src/odin-libraries/go/odinlib"
)

func main() {
    odin, _ := odinlib.Setup("your_config_here.yml")
    resp, _ := http.Get("https://en.wikipedia.org/wiki/Special:Random")
    url := resp.Request.URL.String()
    if odin.Condition("check if url is empty", url != "") {
        odin.Watch("url fetched", url)
        odin.Result("fetch successfully", 0)
    } else {
        odin.Result("failure when fetching", 1)
    }
}
```

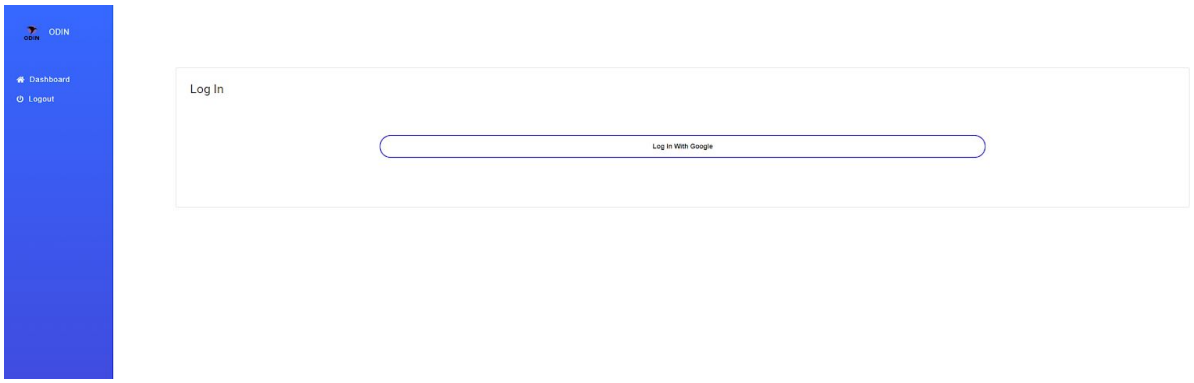
This operation acts like a return statement, once it's executed the job is considered to be over. A successful attempt is denoted by a 0 while a failed attempt is denoted by a 1.

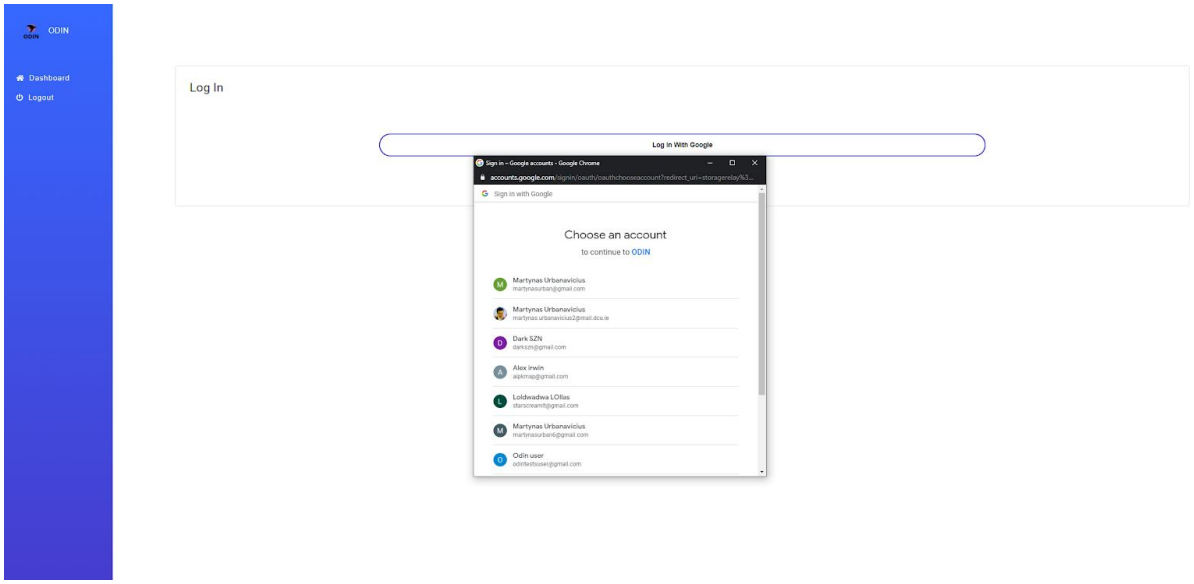
In either case, the result is annotated by a string once more. Once a result operation is run, the code will finish execution.

4.4 The Odin Observability Dashboard

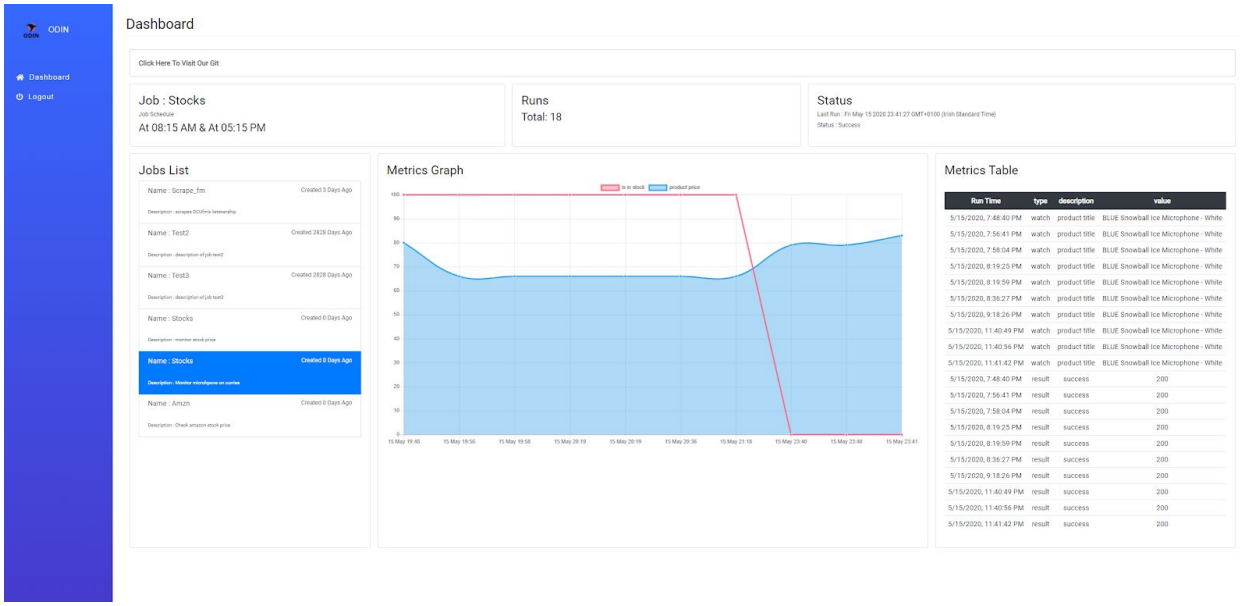
Open Odin Observability Dashboard in browser, if configuration is unchanged this can be done by going to http://localhost:4200

You will be presented with the login page, press log in with google button and follow the google sign in process. If this is your first time logging in, an Odin user account will be made for you automatically.

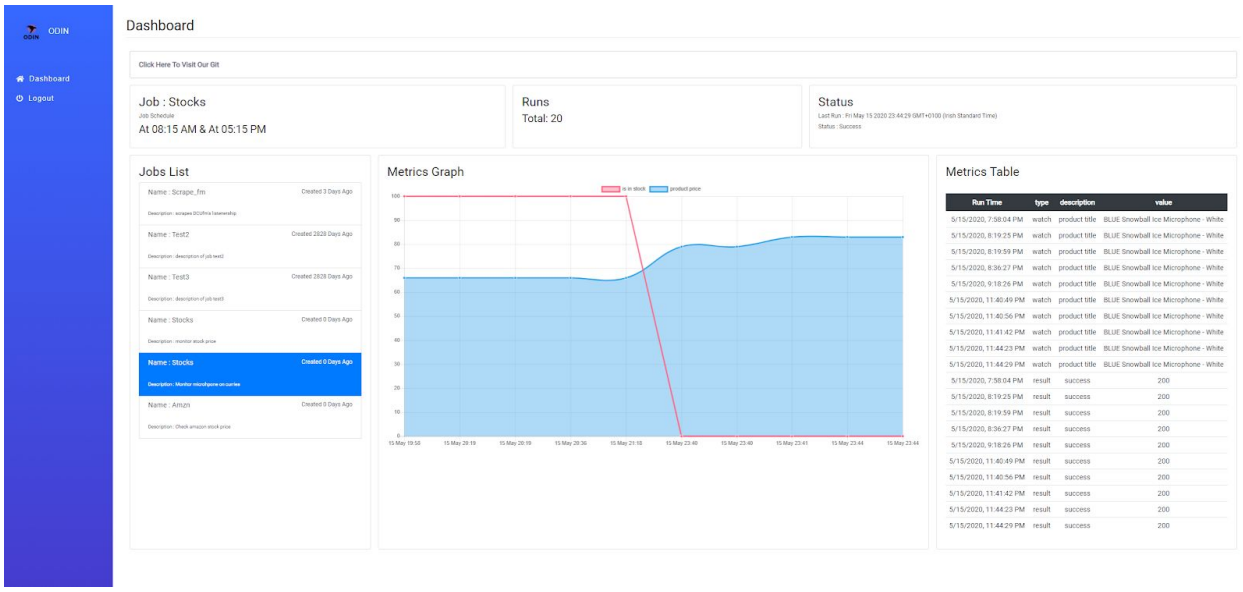




Once logged in you will be presented with the dashboard, a list of your Odin jobs is shown under the ‘Jobs list’ card, this list acts as a menu where you can select a job for which you want to see observability metrics. Upon selecting a different job from the list than the currently highlighted job the dashboard information will update with the metrics of the newly selected job.



After a job is run, the currently selected job metrics will update automatically, as per the difference in metrics cards in between the image above and the image below.



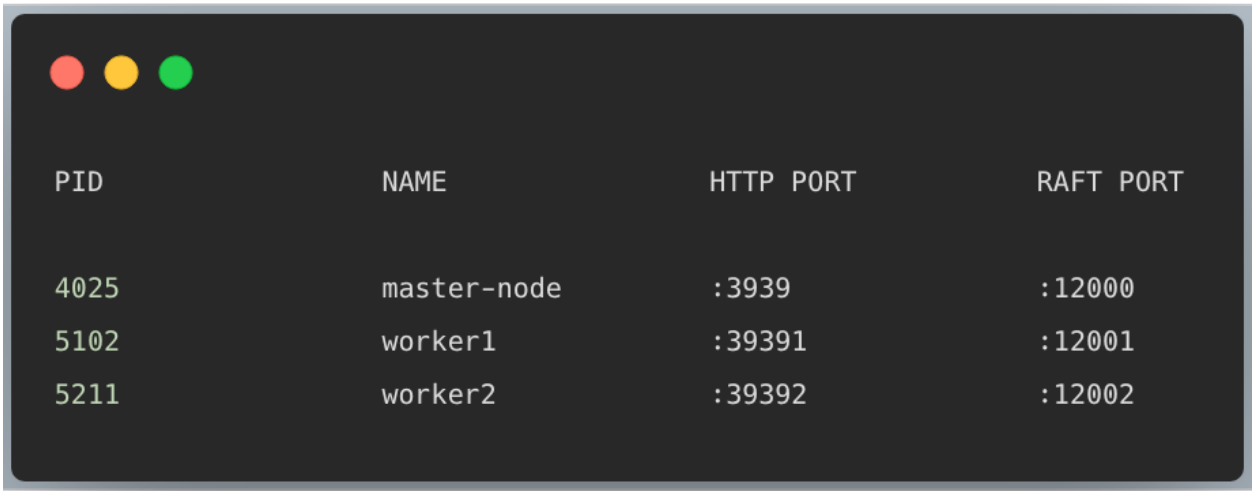
4.5 The Odin Engine as a Distributed System

As previously mentioned Odin Engine leverages the Raft consensus protocol to provide a highly available replicated log to run as a distributed system. Distributed systems offer reliability to systems in case of a failure, and Odin proves to be an easily scalable and flexible system.

Please consult section 4.2.12 in regards to adding nodes to the Odin cluster.

With raft based systems, it's advisable to run 3-node clusters or 5-node clusters. This is recommended as:

- 3-node clusters can tolerate a failure in 1 node
- 5-node clusters can tolerate a failure in any 2 nodes



PID	NAME	HTTP PORT	RAFT PORT
4025	master-node	:3939	:12000
5102	worker1	:39391	:12001
5211	worker2	:39392	:12002

In the example set out in section 4.2.12, if the master node fails, a new master node must be elected. Given the set up in the image, the new master node will be worker1 or worker2. This capacity for failure is one of the greatest benefits with distributed systems. At this point however, you may find that Odin CLI operations will seemingly fail as they are made to interact directly with the master node.

We have built in a `--port` flag for each Odin CLI operation. This allows users to interface with a node of their choice in the case the master node goes offline. As Raft maintains a replicated log between nodes specifying the node to execute upon will not make any difference outside of having to add an additional flag to your commands.

5. Plugins

5.1 Current Plugin Options

In regards to Odin plugins we turn to the various software development kits outlined earlier in the document. Odin jobs are supported in:

- Python
- Go
- Node.js

These software development kits are used to improve the observability of the system, giving you a direct insight into the internal state of jobs.

5.2 Adding Plugins

5.2.1 Python

To add the Python SDK run the following command:

```
pip3 install pyodin
```

From here, please consult with section 4.3.1 in regards to how best utilise the Python SDK

5.2.2 Go

To add the Go SDK run the following command:

```
go get gitlab.computing.dcu.ie/mcdermj7/2020-ca400-urbanam2-mcdermj7/src/odin-libraries/go/odinlib
```

From here, please consult with section 4.3.2 in regards to how best utilise the Go SDK

5.2.3 Node.js

To add the Node.js SDK run the following command:

```
npm install odinlib
```

From here, please consult with section 4.3.2 in regards to how best utilise the Node.js SDK

6. Configuration

6.1 Odin Engine Users

The only users on the system which can avail of Odin resources are those users in the odin Linux group.

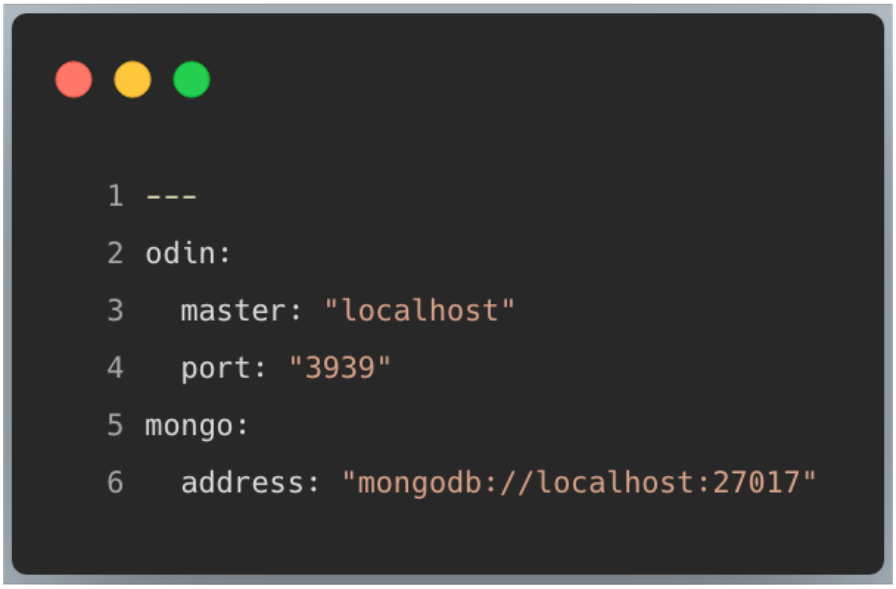
This group can be configured and extended by a root or sudo user on the same system. This can be done using the command:

```
usermod -aG odin <username>
```

Users can only view their own jobs and interact with their own job stats and logs. The root user or sudo users on the system can view all data in the system.

6.2 The Odin Engine Config

When running through section three, the odin-config.yml will be moved to the root user's home directory. Here are the contents of this file:



If you wish to change the default port or address of the Odin Engine, it can be configured here.

6.3 Configuring your Data Store

If at some point you wish to change the MongoDB instance you are using to one which is provided on the cloud, this can be facilitated by just dropping the appropriate link in the mongo.address field of the `odin-config.yml` file.

This alone will not ensure the configuration of your data store location. You must also change the ODIN_MONGODB environment variable:

```
export ODIN_MONGODB="mongodb://new-mongodb-link-here"
```

6.4 Distribution

As outlined elsewhere in the document, you can configure the existing default setup of the Odin Engine to become a distributed system.

With a view to this please consult sections 4.2.12 and 4.5 in the usage guide.